



Programming

SAP Sybase IQ 16.0 SP03

DOCUMENT ID: DC01776-01-1603-01

LAST REVISED: November 2013

Copyright © 2013 by SAP AG or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Please see <http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark> for additional trademark information and notices.

Contents

Partner Certifications	1
Platform Certifications	3
SAP Sybase IQ as a Data Server for Client Applications	5
Open Client Architecture	5
DB-Library and Client Library	5
Network Services	5
Open Client and jConnect Connections	6
login_procedure option	6
Servers with Multiple Databases	9
Using In-Database Analytics in Applications	11
Scalar C or C++ UDF	11
Aggregate C or C++ UDF	11
Java UDFs	12
Java Scalar UDF	12
Java Table UDF	12
Table UDFs	12
TPFs	12
Hadoop Integration	13
Integrating SAP Sybase IQ with a Hadoop Distributed File System	13
Reading a File in a Hadoop Distributed File System as an In-Memory Table	13
Starting an External Hadoop MapReduce Job and Using Results in a Query	15
API Reference for a_v4_extfn	17
Blob (a_v4_extfn_blob)	17
Blob Input Stream (a_v4_extfn_blob_istream)	21
Column Data (a_v4_extfn_column_data)	22
Column List (a_v4_extfn_column_list)	23
Column Order (a_v4_extfn_order_el)	24

Column Subset	
(a_v4_extfn_col_subset_of_input)	25
Describe API	25
Describe Column Type	
(a_v4_extfn_describe_col_type)	90
Describe Parameter Type	
(a_v4_extfn_describe_parm_type)	91
Describe Return (a_v4_extfn_describe_return)	
.....	93
Describe UDF Type	
(a_v4_extfn_describe_udf_type)	95
Execution State (a_v4_extfn_state)	95
External Function (a_v4_extfn_proc)	97
External Procedure Context	
(a_v4_extfn_proc_context)	100
License Information (a_v4_extfn_license_info) ..	112
Optimizer Estimate (a_v4_extfn_estimate)	113
Order By List (a_v4_extfn_orderby_list)	113
Partition By Column Number	
(a_v4_extfn_partitionby_col_num)	114
Row (a_v4_extfn_row)	115
Row Block (a_v4_extfn_row_block)	116
Table (a_v4_extfn_table)	116
Table Context (a_v4_extfn_table_context)	117
Table Functions (a_v4_extfn_table_func)	124
Using SQL in Applications	131
SQL statement execution in applications	131
Prepared statements	132
Prepared Statements Overview	133
Cursor usage	135
Cursors	135
Benefits of using cursors	136
Cursor principles	137
Cursor positioning	138
Cursor behavior when opening cursors	138

Row fetching through a cursor	138
Multiple-row fetching	139
Scrollable cursors	139
Cursors used to modify rows	140
Updatable statements	141
Cursor operations that are canceled	142
Cursor types	142
Availability of cursors	142
Cursor properties	142
Bookmarks and cursors	143
Block cursors	143
SAP Sybase IQ Catalog Store Cursors	143
Catalog Store Cursor Sensitivity	144
Catalog Store Insensitive Cursors	148
Catalog Store Sensitive Cursors	149
Catalog Store Asensitive Cursors	150
Catalog Store Value-Sensitive Cursors	151
Catalog Store Cursor Sensitivity and Performance	153
Catalog Store Cursor Sensitivity and Isolation Levels	157
Requests for SAP Sybase IQ Catalog Store Cursors	158
Result set descriptors	160
Transactions in applications	161
Autocommit and manual commit mode	161
Isolation level settings	163
Cursors and transactions	163
.NET Application Programming	165
SAP Sybase IQ .NET Data Provider	165
SAP Sybase IQ .NET Support	165
SAP Sybase IQ .NET Data Provider Features ...	166
.NET Sample Projects	167
Using the .NET Data Provider in a Visual Studio Project	167

.NET Database Connection Examples	168
Data Access and Manipulation	170
Stored Procedures	185
Transaction Processing	186
Error handling	187
Entity Framework Support	188
SAP Sybase IQ .NET Data Provider Deployment	194
.NET Tracing Support	196
.NET Data Provider Tutorials	200
Tutorial: Using the Simple Code Sample	200
Tutorial: Using the Table Viewer Code Sample .	201
Tutorial: Developing a Simple .NET Database Application with Visual Studio	203
.NET API Reference	211
SAInfoMessageEventHandler(object, SAInfoMessageEventArgs) delegate	211
SARowsCopiedEventHandler(object, SARowsCopiedEventArgs) delegate	211
SARowUpdatedEventHandler(object, SARowUpdatedEventArgs) delegate	212
SARowUpdatingEventHandler(object, SARowUpdatingEventArgs) delegate	212
SABulkCopyOptions() enumeration	212
SAIsolationLevel() enumeration	213
SABulkCopy class	213
SABulkCopyColumnMapping class	216
SABulkCopyColumnMappingCollection class ..	219
DestinationOrdinalComparer class	222
SACommLinksOptionsBuilder class	223
SACommand class	226
SACommandBuilder class	239
SAConnectionStringBuilder class	243
SAConnectionStringBuilderBase class	258
SADDataAdapter class	262

DREnumerator class	270
SADataSourceEnumerator class	271
SADefault class	272
SAError class	273
SAErrorCollection class	275
SAException class	276
SAFactory class	278
SAInfoMessageEventArgs class	282
SAMetaDataCollectionNames class	284
SAParameter class	290
SAParameterCollection class	295
SADBParametersEditor class	301
SAPermission class	302
SAPermissionAttribute class	303
SARowUpdatedEventArgs class	304
SARowUpdatingEventArgs class	305
SARowsCopiedEventArgs class	306
SATcpOptionsBuilder class	308
SATransaction class	315
OLE DB and ADO Development	319
OLE DB	319
Connecting Using OLE DB	320
Supported Platforms	320
Distributed Transactions in OLE DB	320
ADO Programming with SAP Sybase IQ	320
How to Connect to a Database Using the Connection Object	321
How to Execute Statements Using the Command Object	322
How to Obtain Result Sets Using the Recordset Object	323
The Recordset Object	324
Row Updates Through a Cursor Using the Recordset Object	325
ADO Transactions	326

OLE DB Connection Parameters	327
OLE DB Connection Pooling	329
Microsoft Linked Servers	329
Setting up a Linked Server Using an Interactive Application	330
Setting up a Linked Server Using a Script	332
Supported OLE DB Interfaces	333
OLE DB Provider Registration	337
ODBC CLI	339
ODBC conformance	339
ODBC application development	339
ODBC Applications on Windows	340
ODBC applications on Unix	341
The unixODBC driver manager	342
UTF-32 ODBC driver managers for Unix	342
ODBC Samples	343
Building the Sample ODBC Program for Windows	343
Building the Sample ODBC Program for Unix ..	344
ODBC Sample Programs	344
ODBC handles	344
How to allocate ODBC handles	345
ODBC example	346
ODBC Connection Functions	346
Establishing an ODBC Connection	347
Server options changed by ODBC	348
SQLSetConnectAttr extended connection attributes .	349
64-bit ODBC considerations	351
Data alignment requirements	355
Result sets in ODBC applications	356
ODBC transaction isolation levels	356
ODBC cursor characteristics	357
Data retrieval	358
Row updates and deletes through a cursor	360
Bookmarks	360

Stored procedure considerations	361
ODBC escape syntax	362
Error handling in ODBC	365
Java in the Database	369
Java in the Database FAQ	369
What Are the Key Features of Java in the Database?	369
How Can I Use My Own Java Classes in Databases?	369
How Does Java Get Executed in a Database? .	370
Java Error Handling	370
How to Install Java Classes into a Database	371
Class File Creation	371
Special Features of Java Classes in the Database	371
How to Call the Main Method	371
Threads in Java Applications	372
No Such Method Exception	372
How to Return Result Sets from Java Methods .	372
Values Returned from Java Via Stored Procedures	373
Security Management for Java	374
How to Start and Stop the Java VM	374
Shutdown Hooks in the Java VM	374
JDBC CLI	377
JDBC Applications	377
JDBC Drivers	378
JDBC Program Structure	379
Differences Between Client- and Server-Side JDBC Connections	380
SQL Anywhere JDBC Drivers	380
How to Load the SQL Anywhere JDBC 4.0 Driver	381
SQL Anywhere 16 JDBC Driver Connection Strings	381
The jConnect JDBC Driver	382

Installing jConnect System Objects into a Database	382
How to Load the jConnect Driver	383
jConnect Driver Connection Strings	383
Connections from a JDBC Client Application	384
How the Connection Example Works	386
Running the Connection Example	387
How to Establish a Connection from a Server-Side JDBC Class	388
Server-Side Connection Example Code	388
How the Server-Side Connection Example Differs	389
Running the Server-Side Connection Example ..	389
Notes on JDBC Connections	390
Data Access Using JDBC	392
Preparing for the JDBC Examples	392
Inserts, Updates, and Deletes Using JDBC	393
Using Static INSERT and DELETE Statements from JDBC	394
How to Use Prepared Statements for More Efficient Access	395
Using Prepared INSERT and DELETE Statements from JDBC	397
JDBC Batch Methods	398
How to Return Result Sets from Java	399
Returning Result Sets from JDBC	399
JDBC Notes	400
JDBC Callbacks	401
JDBC Escape Syntax	405
JDBC 4.0 API Support	408
Embedded SQL	409
Development Process Overview	410
The SQL Preprocessor	411
Supported Compilers	415
Embedded SQL Header Files	416

Import Libraries	416
Sample Embedded SQL Program	417
Structure of Embedded SQL Programs	418
Loading DBLIB Dynamically Under Windows	418
Sample Embedded SQL Programs	419
Static Cursor Sample	420
Running the Static Cursor Sample Program	420
Dynamic Cursor Sample	421
Running the Dynamic Cursor Sample Program	422
Embedded SQL Data Types	423
Host Variables in Embedded SQL	426
Host Variable Declaration	427
C Host Variable Types	427
Host Variable Usage	431
Indicator Variables	432
The SQL Communication Area (SQLCA)	435
SQLCA Fields	435
SQLCA Management for Multithreaded or Reentrant Code	437
Multiple SQLCAs	439
Static and Dynamic SQL	440
Static SQL Statements	440
Dynamic SQL Statements	440
Dynamic SELECT Statement	442
The SQL Descriptor Area (SQLDA)	443
The SQLDA Header File	443
SQLDA Fields	444
SQLDA Host Variable Descriptions	445
SQLDA sqlLen Field Values	446
How to Fetch Data Using Embedded SQL	452
SELECT Statements That Return at Most One Row	452
Cursors in Embedded SQL	453
Wide Fetches or Array Fetches	456

How to Send and Retrieve Long Values Using Embedded SQL	460
Retrieving LONG Data Using Static SQL	461
Retrieving LONG Data Using Dynamic SQL	462
Sending LONG Data Using Static SQL	462
Sending LONG Data Using Dynamic SQL	463
Simple Stored Procedures in Embedded SQL	463
Stored Procedures with Result Sets	464
Request Management with Embedded SQL	466
Database Backup with Embedded SQL	467
Library Function Reference	467
alloc_sqllda Function	467
alloc_sqllda_noind Function	468
db_backup Function	468
db_cancel_request Function	473
db_change_char_charset Function	473
db_change_nchar_charset Function	474
db_find_engine Function	475
db_fini Function	475
db_get_property Function	476
db_init Function	477
db_is_working Function	477
db_locate_servers Function	478
db_locate_servers_ex Function	479
db_register_a_callback Function	480
db_start_database Function	483
db_start_engine Function	484
db_stop_database Function	485
db_stop_engine Function	485
db_string_connect Function	486
db_string_disconnect Function	487
db_string_ping_server Function	487
db_time_change Function	488
fill_s_sqllda Function	488
fill_sqllda Function	489

fill_sqlda_ex Function	489
free_filled_sqlda Function	490
free_sqlda Function	491
free_sqlda_noinf Function	491
sql_needs_quotes Function	491
sqlda_storage Function	492
sqlda_string_length Function	492
sqlerror_message Function	493
Embedded SQL Statement Summary	493
SAP Sybase IQ Database API for C/C++	495
sqlany_affected_rows(a_sqlany_stmt *) method	495
sqlany_bind_param(a_sqlany_stmt *, sacapi_u32 , a_sqlany_bind_param *) method	495
sqlany_cancel(a_sqlany_connection *) method	496
sqlany_clear_error(a_sqlany_connection *) method	496
sqlany_client_version(char *, size_t) method	496
sqlany_client_version_ex(a_sqlany_interface_conte xt *, char *, size_t) method	497
sqlany_commit(a_sqlany_connection *) method	497
sqlany_connect(a_sqlany_connection *, const char *) method	498
sqlany_describe_bind_param(a_sqlany_stmt *, sacapi_u32 , a_sqlany_bind_param *) method	499
sqlany_disconnect(a_sqlany_connection *) method	499
sqlany_error(a_sqlany_connection *, char *, size_t) method	500
sqlany_execute(a_sqlany_stmt *) method	500
sqlany_execute_direct(a_sqlany_connection *, const char *) method	501
sqlany_execute_immediate(a_sqlany_connection *, const char *) method	502
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) method	503

sqlany_fetch_next(a_sqlany_stmt *) method503
 sqlany_finalize_interface(SQLAnywhereInterface *)
 method504
 sqlany_fini() method505
 sqlany_fini_ex(a_sqlany_interface_context *) method
 505
 sqlany_free_connection(a_sqlany_connection *)
 method505
 sqlany_free_stmt(a_sqlany_stmt *) method506
 sqlany_get_bind_param_info(a_sqlany_stmt *,
 sacapi_u32 , a_sqlany_bind_param_info *) method
 506
 sqlany_get_column(a_sqlany_stmt * , sacapi_u32 ,
 a_sqlany_data_value *) method507
 sqlany_get_column_info(a_sqlany_stmt * ,
 sacapi_u32 , a_sqlany_column_info *) method508
 sqlany_get_data(a_sqlany_stmt * , sacapi_u32 ,
 size_t , void * , size_t) method508
 sqlany_get_data_info(a_sqlany_stmt * , sacapi_u32 ,
 a_sqlany_data_info *) method509
 sqlany_get_next_result(a_sqlany_stmt *) method509
 sqlany_init(const char * , sacapi_u32 , sacapi_u32 *)
 method510
 sqlany_init_ex(const char * , sacapi_u32 , sacapi_u32
 *) method511
 sqlany_initialize_interface(SQLAnywhereInterface * ,
 const char *) method511
 sqlany_make_connection(void *) method512
 sqlany_make_connection_ex(a_sqlany_interface_co
 ntext * , void *) method512
 sqlany_new_connection(void) method513
 sqlany_new_connection_ex(a_sqlany_interface_con
 text *) method513
 sqlany_num_cols(a_sqlany_stmt *) method514
 sqlany_num_params(a_sqlany_stmt *) method514

sqlany_num_rows(a_sqlany_stmt *) method	514
sqlany_prepare(a_sqlany_connection *, const char *) method	515
sqlany_reset(a_sqlany_stmt *) method	516
sqlany_rollback(a_sqlany_connection *) method	516
sqlany_send_param_data(a_sqlany_stmt *, sacapi_u32 , char *, size_t) method	517
sqlany_sqlstate(a_sqlany_connection *, char *, size_t) method	517
a_sqlany_data_direction() enumeration	518
a_sqlany_data_type() enumeration	518
a_sqlany_native_type() enumeration	519
SACAPI_ERROR_SIZE variable	520
SQLANY_API_VERSION_1 variable	520
SQLANY_API_VERSION_2 variable	520
SQLAnywhereInterface structure	520
dll_handle void *	521
initialized int	521
sqlany_affected_rows void *	521
sqlany_bind_param void *	521
sqlany_cancel void *	521
sqlany_clear_error void *	521
sqlany_client_version void *	522
sqlany_client_version_ex void *	522
sqlany_commit void *	522
sqlany_connect void *	522
sqlany_describe_bind_param void *	522
sqlany_disconnect void *	522
sqlany_error void *	523
sqlany_execute void *	523
sqlany_execute_direct void *	523
sqlany_execute_immediate void *	523
sqlany_fetch_absolute void *	523
sqlany_fetch_next void *	523
sqlany_fini void *	524

sqlany_fini_ex void *	524
sqlany_free_connection void *	524
sqlany_free_stmt void *	524
sqlany_get_bind_param_info void *	524
sqlany_get_column void *	524
sqlany_get_column_info void *	525
sqlany_get_data void *	525
sqlany_get_data_info void *	525
sqlany_get_next_result void *	525
sqlany_init void *	525
sqlany_init_ex void *	525
sqlany_make_connection void *	526
sqlany_make_connection_ex void *	526
sqlany_new_connection void *	526
sqlany_new_connection_ex void *	526
sqlany_num_cols void *	526
sqlany_num_params void *	526
sqlany_num_rows void *	527
sqlany_prepare void *	527
sqlany_reset void *	527
sqlany_rollback void *	527
sqlany_send_param_data void *	527
sqlany_sqlstate void *	527
a_sqlany_bind_param structure	528
direction a_sqlany_data_direction	528
name char *	528
value a_sqlany_data_value	528
a_sqlany_bind_param_info structure	528
direction a_sqlany_data_direction	529
input_value a_sqlany_data_value	529
name char *	529
output_value a_sqlany_data_value	529
a_sqlany_column_info structure	529
max_size size_t	530
name char *	530

native_type a_sqlany_native_type	530
nullable sacapi_bool	530
precision unsigned short	530
scale unsigned short	531
type a_sqlany_data_type	531
a_sqlany_data_info structure	531
data_size size_t	531
is_null sacapi_bool	531
type a_sqlany_data_type	532
a_sqlany_data_value structure	532
buffer char *	532
buffer_size size_t	532
is_null sacapi_bool *	533
length size_t *	533
type a_sqlany_data_type	533
Perl DBI Support	535
DBD::SQLAnywhere	535
Installing DBD::SQLAnywhere on Windows	535
Installing DBD::SQLAnywhere on Unix	537
Perl Scripts That Use DBD::SQLAnywhere	538
The DBI Module	538
How to Open and Close a Database Connection Using Perl DBI	539
How to Obtain Result Sets Using Perl DBI	540
How to Process Multiple Result Sets Using Perl DBI	541
How to Insert Rows Using Perl DBI	542
Python Support	545
sqlanydb	545
Installing Python Support on Windows	546
Installing Python Support on Unix	546
Python Scripts That Use sqlanydb	547
The sqlanydb Module	547
How to Open and Close a Database Connection Using Python	548

How to Obtain Result Sets Using Python	548
How to Insert Rows Using Python	549
Database Type Conversion	550
PHP Support	553
SAP Sybase IQ PHP Extension	553
Testing the PHP Extension	553
Creating and Running PHP Test Pages	554
PHP Script Development	556
How to Build the SAP Sybase IQ PHP Extension on Unix	561
SAP Sybase IQ PHP API Reference	566
sasql_affected_rows	566
sasql_commit	567
sasql_close	567
sasql_connect	567
sasql_data_seek	568
sasql_disconnect	568
sasql_error	568
sasql_errorcode	569
sasql_escape_string	569
sasql_fetch_array	570
sasql_fetch_assoc	570
sasql_fetch_field	571
sasql_fetch_object	571
sasql_fetch_row	572
sasql_field_count	572
sasql_field_seek	572
sasql_free_result	573
sasql_get_client_info	573
sasql_insert_id	573
sasql_message	574
sasql_multi_query	574
sasql_next_result	575
sasql_num_fields	575
sasql_num_rows	575

sasql_pconnect	576
sasql_prepare	576
sasql_query	576
sasql_real_escape_string	577
sasql_real_query	577
sasql_result_all	578
sasql_rollback	579
sasql_set_option	579
sasql_stmt_affected_rows	580
sasql_stmt_bind_param	580
sasql_stmt_bind_param_ex	581
sasql_stmt_bind_result	582
sasql_stmt_close	582
sasql_stmt_data_seek	582
sasql_stmt_errno	583
sasql_stmt_error	583
sasql_stmt_execute	583
sasql_stmt_fetch	584
sasql_stmt_field_count	584
sasql_stmt_free_result	584
sasql_stmt_insert_id	585
sasql_stmt_next_result	585
sasql_stmt_num_rows	586
sasql_stmt_param_count	586
sasql_stmt_reset	586
sasql_stmt_result_metadata	587
sasql_stmt_send_long_data	587
sasql_stmt_store_result	587
sasql_store_result	588
sasql_sqlstate	588
sasql_use_result	589
Ruby Support	591
Ruby API Support	591
Configuring Rails Support in SAP Sybase IQ ...	592
Ruby-DBI Driver	595

SAP Sybase IQ Ruby API Reference	599
sqlany_affected_rows	600
sqlany_bind_param Function	600
sqlany_clear_error Function	601
sqlany_client_version Function	601
sqlany_commit Function	601
sqlany_connect Function	602
sqlany_describe_bind_param Function	602
sqlany_disconnect Function	603
sqlany_error Function	603
sqlany_execute Function	604
sqlany_execute_direct Function	604
sqlany_execute_immediate Function	605
sqlany_fetch_absolute Function	605
sqlany_fetch_next Function	606
sqlany_fini Function	607
sqlany_free_connection Function	607
sqlany_free_stmt Function	608
sqlany_get_bind_param_info Function	608
sqlany_get_column Function	609
sqlany_get_column_info Function	609
sqlany_get_next_result Function	610
sqlany_init Function	611
sqlany_new_connection Function	611
sqlany_num_cols Function	612
sqlany_num_params Function	612
sqlany_num_rows Function	612
sqlany_prepare Function	613
sqlany_rollback Function	614
sqlany_sqlstate Function	614
Column Types	614
Native Column Types	615
Sybase Open Client Support	617
Open Client Architecture	617
What You Need to Build Open Client Applications	618

Open Client Data Type Mappings	619
Range Limitations in Open Client Data Type Mapping	619
SQL in Open Client Applications	620
Open Client SQL Statement Execution	620
Open Client Prepared Statements	621
Open Client Cursor Management	621
Open Client Result Sets	622
Known Open Client Limitations of SAP Sybase IQ	623
HTTP Web Services	625
SAP Sybase IQ As an HTTP Web Server	625
Quick Start to Using SAP Sybase IQ As an HTTP Web Server	625
How to Start an HTTP Web Server	626
What Are Web Services	629
How to Develop Web Service Applications in an HTTP Web Server	638
How to Browse the SAP Sybase IQ HTTP Web Server	654
Access to Web Services Using Web Clients	658
Quick Start to Using SAP Sybase IQ As a Web Client	658
Quick Start to Accessing an SAP Sybase IQ HTTP Web Server	660
Web Client Application Development	662
HTTP and SOAP Request Structures	693
How to Log Web Client Requests	694
Web Services References	695
Web Service Error Code Reference	695
HTTP Web Service Examples	697
Tutorial: Create a Web Server and Access It from a Web Client	697
Tutorial: Using SAP Sybase IQ to Access a SOAP/DISH Service	701

Tutorial: Using Visual C# to Access a SOAP/ DISH Web Service	709
Tutorial: Using JAX-WS to Access a SOAP/ DISH Web Service	715
Three-Tier Computing and Distributed Transactions	725
Three-Tier Computing Architecture	725
Distributed Transactions in Three-Tier Computing	726
The Vocabulary of Distributed Transactions	727
How Application Servers Use DTC	727
Distributed Transaction Architecture	728
Distributed Transactions	728
DTC Isolation Levels	729
Recovery From Distributed Transactions	729
Database Tools Interface (DBTools)	731
DBTools Import Libraries	732
DBTools Library Initialization and Finalization	732
DBTools Function Calls	733
Callback Functions	733
Version Numbers and Compatibility	735
Bit Fields	735
A DBTools Example	736
Software Component Exit Codes	738
Database Tools C API Reference	739
DBBackup(const a_backup_db *) method	740
DBChangeLogName(const a_change_log *) method	740
DBCCreate(a_create_db *) method	741
DBCCreatedVersion(a_db_version_info *) method	741
DBErase(const an_erase_db *) method	741
DBInfo(a_db_info *) method	742
DBInfoDump(a_db_info *) method	742
DBInfoFree(a_db_info *) method	743
DBLicense(const a_dblic_info *) method	743

DBLogFileInfo(const a_log_file_info *) method	743
DBRemoteSQL(a_remote_sql *) method	744
DBSynchronizeLog(const a_sync_db *) method	744
DBToolsFini(const a_dbtools_info *) method	744
DBToolsInit(const a_dbtools_info *) method	745
DBToolsVersion(void) method	745
DBTranslateLog(const a_translate_log *) method	746
DBTruncateLog(const a_truncate_log *) method	746
DBUnload(an_unload_db *) method	746
DBUpgrade(const an_upgrade_db *) method	747
DBValidate(const a_validate_db *) method	747
Autotune() enumeration	748
Checkpoint() enumeration	748
History() enumeration	748
Padding() enumeration	749
Unit() enumeration	749
Unload() enumeration	749
UserList() enumeration	749
Validation() enumeration	750
Verbosity() enumeration	750
Version() enumeration	750
a_backup_db structure	751
a_change_log structure	756
a_create_db structure	759
a_db_info structure	765
a_db_version_info structure	770
a_dblic_info structure	771
a_dbtools_info structure	773
a_log_file_info structure	773
a_name structure	775

a_remote_sql structure	775
a_sync_db structure	787
a_syncpub structure	808
a_sysinfo structure	809
a_table_info structure	810
a_translate_log structure	812
a_truncate_log structure	821
a_validate_db structure	822
an_erase_db structure	824
an_unload_db structure	826
an_upgrade_db structure	839
Appendix: Using OLAP	843
About OLAP	843
OLAP Benefits	844
OLAP Evaluation	844
GROUP BY Clause Extensions	845
Group by ROLLUP and CUBE	846
Analytical Functions	858
Simple Aggregate Functions	859
Windowing	859
Numeric Functions	892
OLAP Rules and Restrictions	901
Additional OLAP Examples	902
Example: Window Functions in Queries	903
Example: Window With Multiple Functions	904
Example: Calculate Cumulative Sum	904
Example: Calculate Moving Average	905
Example: ORDER BY Results	905
Example: Multiple Aggregate Functions in a Query	906
Example: Window Frame Comparing ROWS and RANGE	906
Example: Window Frame Excludes Current Row	907
Example: Window Frame for RANGE	908

Example: Unbounded Preceding and Unbounded Following	908
Example: Default Window Frame for RANGE	909
BNF Grammar for OLAP Functions	910
Appendix: Accessing Remote Data	917
SAP Sybase IQ and Remote Data	917
Characteristics of Sybase Open Client and jConnect connections	917
Requirements for Accessing Remote Data	919
Remote Servers	940
External Logins	948
Proxy tables	948
Joins between remote tables	951
Joins between tables from multiple local databases	952
Native statements and remote servers	953
Remote Procedure Calls (RPCs)	953
Remote Transactions	954
Remote transaction management	954
Remote Transaction Restrictions	955
Internal Operations	955
Query Parsing	955
Query Normalization	955
Query preprocessing	955
Complete passthrough of the statement	956
Partial passthrough of the statement	956
Remote Data Access Troubleshooting	957
Features not supported for remote data	958
Case sensitivity	958
Connectivity tests	958
Remote data access connections via ODBC	959
Remote data access on multiplex servers	959
Appendix: SQL Reference	961
ALTER SERVER Statement	961
CREATE EXISTING TABLE Statement	964

Contents

CREATE SERVER Statement	966
CREATE TABLE Statement	968
DROP SERVER Statement	985
Index	987

Partner Certifications

The SAP® Sybase® IQ partner ecosystem includes certified partners, data warehouse infrastructure partners, analytics solutions partners, and business intelligence partner applications.

For certification reports and the list of SAP Sybase IQ partners, see the *SAP Sybase IQ Marketplace*.

Partner Certifications

Platform Certifications

Certified means that a product runs on, and is supported on, a specific platform environment. SAP Sybase IQ is certified on certain operating systems with specific CPU architecture combinations.

Find certified product-platform combinations at <http://certification.sybase.com/ucr/search.do>.

SAP Sybase IQ as a Data Server for Client Applications

SAP Sybase IQ supports client application connections through either ODBC or JDBC. Use SAP Sybase IQ as a data server for client applications.

With certain limitations, SAP Sybase IQ may also appear to certain client applications as an Open Server™.

The facilities described in this chapter do not provide remote data access for IQ users on Windows and Sun Solaris systems. Remote data access is provided by Component Integration Services (CIS), the core interoperability feature of Enterprise Connect™ Data Access (ECDA).

Open Client Architecture

The primary documentation for Sybase Open Client™ application development is the Open Client documentation, available from SAP. This section describes features specific to SAP Sybase IQ, but it is not an exhaustive guide to Sybase Open Client application programming.

Sybase Open Client has two components: programming interfaces and network services.

DB-Library and Client Library

Sybase Open Client provides two core programming interfaces for writing client applications: DB-Library™ and Client-Library.

Open Client DB-Library provides support for older Open Client applications, and is a completely separate programming interface from Client-Library. DB-Library is documented in the *Open Client DB-Library/C Reference Manual*, provided with the Sybase Open Client product.

Client-Library programs also depend on CS-Library, which provides routines that are used in both Client-Library and Server-Library applications. Client-Library applications can also use routines from Bulk-Library to help high-speed data transfer.

Both CS-Library and Bulk-Library are included in the Sybase Open Client, which is available separately.

Network Services

Open Client network services include Sybase Net-Library, which provides support for specific network protocols such as TCP/IP and DECnet. The Net-Library interface is invisible to application developers. However, on some platforms, an application may need a different Net-Library driver for different system network configurations. Depending on your host

platform, the Net-Library driver is specified either by the system's Sybase configuration or when you compile and link your programs.

Instructions for driver configuration can be found in the *Open Client/Server Configuration Guide*.

Instructions for building Client-Library programs can be found in the *Open Client/Server Programmer's Supplement*.

Open Client and jConnect Connections

When SAP Sybase IQ serves applications over TDS, it automatically sets relevant database options to values that are compatible with SAP Sybase SQL Anywhere® Server default behavior. These options are set temporarily, for the duration of the connection only. The client application can override these options at any time.

Note: SAP Sybase IQ does not support the `ANSI_BLANKS`, `FLOAT_AS_DOUBLE`, and `TSQL_HEX_CONSTANT` options.

Although SAP Sybase IQ allows longer user names and passwords, TDS client user names and passwords cannot exceed 30 bytes. If your password or user ID is longer than 30 bytes, attempts to connect over TDS (for example, using jConnect) return an `Invalid user ID or password` error.

Note: ODBC applications, including Interactive SQL applications, automatically set certain database options to values mandated by the ODBC specification. This overwrites settings by the `LOGIN_PROCEDURE` database option.

login_procedure option

Specifies a login procedure that sets connection compatibility options at startup.

Allowed values

String

Default

sp_login_environment system procedure

Scope

Can be set for an individual connection or for PUBLIC. You must have the SET ANY SECURITY OPTION system privilege to set this option.

Remarks

This login procedure calls the sp_login_environment procedure at run time to determine the database connection settings. The login procedure is called after all the checks have been performed to verify that the connection is valid. The procedure specified by the

login_procedure option is not executed for event connections, but it is executed for web service connections.

You can customize the default database option settings by creating a new procedure and setting login_procedure to call the new procedure. This custom procedure needs to call either sp_login_environment or detect when a TDS connection occurs (see the default sp_login_environment code) and call sp_tsq_l_environment directly. Failure to do so can break TDS-based connections. Do not edit either sp_login_environment or sp_tsq_l_environment.

A password expired error message with SQLSTATE 08WA0 can be signaled by a user-defined login procedure to indicate to a user that their password has expired. Signaling the error allows applications to check for the error and process expired passwords. It is recommended that you use a login policy to implement password expiry and not a login procedure that returns the expired password error message.

If you use the NewPassword=* connection parameter, signaling this error is required for the client libraries to prompt for a new password. If the procedure signals SQLSTATE 28000 (invalid user ID or password) or SQLSTATE 08WA0 (expired password), or the procedure raises an error with RAISERROR, the login fails and an error is returned to the user. If you signal any other error or if another error occurs, then the user login is successful and a message is written to the database server message log.

Example

The following example shows how you can disallow a connection by signaling the INVALID_LOGON error.

```
CREATE PROCEDURE DBA.login_check ( )
  BEGIN
    DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
    // Allow a maximum of 3 concurrent connections
    IF( DB_PROPERTY( 'ConnCount' ) > 3 ) THEN
      SIGNAL INVALID_LOGON;
    ELSE
      CALL sp_login_environment;
    END IF;
  END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

The following example shows how you can block connection attempts if the number of failed connections for a user exceeds 3 within a 30 minute period. All blocked attempts during the block out period receive an invalid password error and are logged as failures. The log is kept long enough for a DBA to analyze it.

```
CREATE TABLE DBA.ConnectionFailure (
  pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
```

SAP Sybase IQ as a Data Server for Client Applications

```
    user_name CHAR(128) NOT NULL,
    tm TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
)
go

CREATE INDEX ConnFailTime ON DBA.ConnectionFailure(
    user_name, tm )
go

CREATE EVENT ConnFail TYPE ConnectFailed
HANDLER
BEGIN
    DECLARE usr CHAR(128);
    SET usr = event_parameter( 'User' );

    // Put a limit on the number of failures logged.
    IF (SELECT COUNT(*) FROM DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT_TIMESTAMP )) < 20 THEN
        INSERT INTO DBA.ConnectionFailure( user_name )
            VALUES( usr );
        COMMIT;
        // Delete failures older than 7 days.
        DELETE DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm < dateadd( day, -7, CURRENT_TIMESTAMP );
        COMMIT;
    END IF;
END
go

CREATE PROCEDURE DBA.login_check( )
BEGIN
    DECLARE usr CHAR(128);
    DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
    SET usr = CONNECTION_PROPERTY( 'Userid' );
    // Block connection attempts from this user
    // if 3 or more failed connection attempts have occurred
    // within the past 30 minutes.
    IF ( SELECT COUNT( * ) FROM DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT_TIMESTAMP ) ) >= 3 THEN
        SIGNAL INVALID_LOGON;
    ELSE
        CALL sp_login_environment;
    END IF;
END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

The following example shows how to signal an error indicating that the user's password has expired. It is recommended that you use a login policy to implement password expiry notification.

```
CREATE PROCEDURE DBA.check_expired_login( )
BEGIN
  DECLARE PASSWORD_EXPIRED EXCEPTION FOR SQLSTATE '08WA0';

  IF( condition-to-check-for-expired-password ) THEN
    SIGNAL PASSWORD_EXPIRED;
  ELSE
    CALL sp_login_environment;
  END IF;
END;
```

Servers with Multiple Databases

Using Open Client Library, you can connect to a specific database on a server containing multiple databases.

- Set up entries in the `interfaces` file for the server.
- Use the `-n` parameter on the `start_iq` command to set up a shortcut for the database name.
- Specify the `-S database_name` parameter with the database name on the `isql` command. This parameter is required whenever you connect.

You can run the same program against multiple databases without changing the program itself by putting the shortcut name into the program and merely changing the shortcut definition.

For example, the following `interfaces` file excerpt defines two servers, `live_sales` and `test_sales`:

```
live_sales
```

```
  query tcp ether myhostname 5555
  master tcp ether myhostname 5555
```

```
test_sales
```

```
  query tcp ether myhostname 7777
  master tcp ether myhostname 7777
```

Start the server and set up an alias for a particular database. The following command sets `live_sales` equivalent to `salesbase.db`:

```
start_iq -n sales_live <other parameters> -x \ 'tcpip{port=5555}'
salesbase.db -n live_sales
```

To connect to the `live_sales` server:

```
isql -Udba -Psql -Slive_sales
```

A server name may only appear once in the `interfaces` file. Because the connection to SAP Sybase IQ is now based on the database name, the database name must be unique. If all

SAP Sybase IQ as a Data Server for Client Applications

your scripts are set up to work on salesbase database, you will not have to modify them to work with live_sales or test_sales.

Using In-Database Analytics in Applications

SAP Sybase IQ provides in-database analytics in three ways: native built-in analytics, native UDF plug-in analytics, and external UDF plug-in analytics. As a developer, you can enable complex analysis of big data by providing analytics as external UDFs.

- **Native built-in analytics** – Examples of native in-kernel analytics include OLAP, and full-text search. The **CUME_DIST** function is one example of a built-in ANSI SQL OLAP built-in aggregate function.
- **Native UDF plug-in analytics** – Using out-of-process shared libraries, you can develop *text analytics* solutions. By developing out-of-process in-database UDFs, you minimize the security and robustness risks inherent in running user-defined code in-process. See *Unstructured Data Analytics* for LOB documentation.
- **External UDF plug-in analytics** – Use Java UDFs, table UDFs, and Table Parameterized Functions (TPFs) to develop out-of-process analytics solutions for big data.

See also

- *Appendix: Using OLAP* on page 843

Scalar C or C++ UDF

A scalar UDF is a V3 or V4 external C or C++ procedure that operates on a single value.

See User-Defined Functions for detailed information and examples. External C and C++ procedures require a separately licensed SAP Sybase IQ option.

Aggregate C or C++ UDF

An aggregate UDF is a V3 or V4 external C or C++ procedure that operates on multiple values. Aggregate UDFs are also sometimes known as UDAs or UDAFs. The context structure for coding aggregate UDFs is slightly different than the context structure used for coding scalar UDFs.

See User-Defined Functions for detailed information and examples. External C and C++ procedures require a separately licensed SAP Sybase IQ option.

Java UDFs

Java UDFs behave like SQL functions except that the code for the procedure or function is written in Java, and the execution takes place outside the database server, within a Java VM environment. You can define Java scalar UDFs, and Java table UDFs.

Java UDFs do not require a separately licensed SAP Sybase IQ option.

Java Scalar UDF

An out-of-process (external environment) scalar user-defined function implemented in Java code.

See *User-Defined Functions* for detailed information and examples.

Java Table UDF

An out-of-process (external environment) table UDF implemented in Java code.

See *User-Defined Functions* for detailed information and examples.

Table UDFs

Table UDFs are external user-defined C, C++, or Java table functions. Unlike scalar and aggregate UDFs, table UDFs produce row sets as output. A SQL query can consume the row sets as a table expression in the FROM clause of a SQL statement.

Scalar and aggregate UDFs can use either the v3 or v4 extfn API, but table UDFs can use only v4.

See *User-Defined Functions* for detailed information and examples.

TPFs

Table parameterized functions (TPFs) are enhanced table UDFs that accept either scalar values or row sets as input. You can configure user-specified partitioning for your TPF. The UDF developer can declare a partitioning scheme that breaks down the dataset into smaller pieces of query processing that you distribute across multiplex nodes. This enables you to execute the TPF in parallel in a distributed server environment over partitions of row sets. The query engine supports massive parallelization of TPF processing.

See *User-Defined Functions* for detailed information and examples.

Hadoop Integration

SAP Sybase IQ includes a UDF API that you can use to build MapReduce components, which can be used for Hadoop integration. The SAP Sybase solutions store has examples of Hadoop integration.

The MapReduce programming model is designed for massively parallel distributed computing. The MapReduce programming model consists of two main stages:

- **Map stage** – The leader node divides a problem into subproblems or *maps*. These maps must be independent of each other and are executed in parallel.
- **Reduce stage** – The leader node collects the answers of the subproblems and combines them in a meaningful way to get the answer to the original problem.

Apache Hadoop is a MapReduce implementation. Hadoop is a Java software framework that automates scheduling of map and reduce jobs.

SAP Sybase IQ supports Hadoop-like parallel scheduling using Table Parameterized Functions (TPFs), a class of external user-defined functions. TPFs accept arbitrary rowsets of table-valued input parameters, and can be parallelized in a distributed server environment. You can specify partitioning and ordering requirements on the TPF input. As a developer, you can use TPFs to exploit the MapReduce paradigm from within the database server, using SQL.

For TPF fundamentals, see the *User-Defined Functions* guide.

Integrating SAP Sybase IQ with a Hadoop Distributed File System

The data returned from a Hadoop analysis can be integrated into an SAP Sybase IQ database in several ways.

- **ETL Processing** – Bulk load data from Hadoop data stores into SAP Sybase IQ using the open source utility SCOOP.
- **Data Federation** – Expose HDFS files as tables in an SAP Sybase IQ database that participate in SQL queries. The HDFS files do not need to be loaded into SAP Sybase IQ.
- **Query Federation** – Allow SQL queries in SAP Sybase IQ to execute Hadoop processes that return data that is incorporated into the SQL result set.
- **Client-side Federation** – Federate queries across SAP Sybase IQ databases and Hadoop files using the TOAD™ SQL tool.

Reading a File in a Hadoop Distributed File System as an In-Memory Table

A data federation example where SAP Sybase IQ reads a file in the Hadoop Distributed File System (HDFS) as an in-memory table.

Note: This sample code is primarily for illustration purposes and is not intended for production. Although effort was made to ensure reasonable error handling, the examples are

not production-grade and will require additional safeguards and testing prior to using in production.

1. Create the Java class:

```
public class HDFSClient {
    public static void readFileByLine(String file, ResultSet
rset[])
throws IOException {

// Set Configuration to point to HDFS NameNode and find input dir
Configuration conf = new Configuration();
conf.addResource(new Path("/home/mymachine/hadoop/conf/core-
site.xml"));
FileSystem fileSystem = FileSystem.get(conf);
Path path = new Path(file);
if (!fileSystem.exists(path)); {
    System.out.println("File " + file + " does not exists");
    return;
}

// Create meta data for the result set
ResultSetMetaDataImpl rsmd = new ResultSetMetaDataImpl(1);
rsmd.setColumnType(1, Typs.VARCHAR);
rsmd.setColumnname(1, "c1");
rsmd.setColumnLabel(1, "c1");
rsmd.setColumnDisplaySize(1, "c1");
rsmd.setTableName(1, "MyTable");

// Create ResultSet using the meta data
ResultSetImpl rs = null;
try {
    rs = new ResultSetImpl((ResultSetMetaDataImpl)rsmd);
    rs.beforeFirst();// Make sure we are at the beginning
} catch(Exception e) {
    System.out.println("Could not create result set.");
    System.out.println(e.toString());
}

// Read files from input dir line by line inserting into rs
String line;
DataInputStream in = new DataInputSteam(fileSystem.open(path));
BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
while ((line = reader.readLine()) != null) {
try {
    rs.insertRow();// Insert a new row
    rs.updateString(1, (line));
} catch(Exception e) {
    System.out.println("Could not insert row/data");
    System.out.println(e.toString());
}
}
try {
rs.beforeFirst();// Make sure we are at the beginning
} catch(Exception e) {
```



```

        System.out.println(e.toString());
    }

    rset[0] = rs; // Assign result set to the 1st of the passé din
    array.

    in.close();
    reader.close();
    fileSystem.close();

}
}
}

```

2. Install the class or the packaged JAR file:

```
INSTALL JAVA NEW JAR 'myjar' FROM FILE '/home/mymachine/UDFs/
myjar.jar';
```

3. Create the function:

```
CREATE or REPLACE PROCEDURE readFileByLine( IN fileName CHAR(50) )
RESULT ( c1 VARCHAR(255) )
EXTERNAL NAME 'example.HDFSclient.readFileByLine(Ljava/lang/
String; [Ljava/sql/ResultSet;)V'
LANGUAGE JAVA;
```

4. Execute the function:

```
SELECT c1 FROM readFileByLine('/home/mymachine/input/input.txt');
```

Starting an External Hadoop MapReduce Job and Using Results in a Query

Define the map and reduce methods to input and output data structured in <key, value> pairs.

Assume you have a directory with two text files with the following contents:

- File1.txt: Hello World Goodbye World
- File2.txt: Goodbye World Hadoop

1. During the mapping step, each file is worked on as a separate map job and the output from each of these maps is the following <key, value>:

- Job1: <Hello, one> <World, one> <Goodbye, one> <World, one>
- Job2: <Goodbye, one> <world, one> <Hadoop, one>

2. Call the Reducer which simply adds the <key, value> outputted from the Map step. Output from the local Reducer is:

- Job1: <Hello, one> <World, two> <Goodbye, one>
- Job2: <Goodbye, one> <World, one> <Hadoop, one>

3. Combine to get the final output:

```
<Hello, one> <World, 3><Goodbye, 2><Hadoop, 1>
```

Note: This sample code is primarily for illustration purposes and is not intended for production. Although effort was made to ensure reasonable error handling, the examples are

not production-grade and will require additional safeguards and testing prior to using in production.

```
public class WordCountDriver extends Configured {
    public static void String HADOOP_ROOT_DIR = "hdfs://localhost:
9000"
private Text word = new Text();
private final IntWritable one = new IntWritable(1);

static class WordCountMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {

public void map(LongWritable key Text value, Context context) throws
IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line.toLowerCase());
    while (itr.hasMoreTokens()){
        word.set(itr.next(Token));
        context.write(word, one);
    }
};

static class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable > {

public void reduce (Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable value : values) {
        sum += value.get();
    }
    context.write(key, new IntWritable(sum));
}
};

Public static void run(String input, String output, ResultSet rs[])
throws Exception {
    Configuration conf = new Configuration();
    conf.addResource(new Path("/home/mymachine/hadoop/conf/core-
site.xml"));
    conf.set("fs.default.name", "hdfs://localhost:9000");
    conf.set("mapred.job.tracker", "localhost:9000");

    // Specify output types
    Job job = new Job(conf, "Word Count");
    Job.setOutputKeyClass(Text.class);
    Job.setOutputValueClass(IntWritable.class);

    // Specify input and output locations
    FileInputFormat.addInputPath(job, new Path(HADOOP_ROOT_DIR+input));
    FileOutputFormat.addInputPath(job, new Path(HADOOP_ROOT_DIR
+output));

    // Specify a mapper
    job.setMapperClass(WordCountDriver.WordCountMapper.class);
```

```

// Specify a reducer
job.setReducerClass(WordCountDriver.WordCountReducer.class);
job.setCombinerClass(WordCountDriver.WordCountReducer.class);
job.setJarByClass(WordCountDriver.class)

// Wait for MR job to complete
while (job.waitForCompletion(true) ? false : true) {
    // Waiting..
}

HDFSClient hdfsc = new HDFSClient();
hdfsc.readFileByLine(file, rs);
}
}
}

```

API Reference for a_v4_extfn

Reference information for a_v4_extfn functions, methods, and attributes.

Blob (a_v4_extfn_blob)

Use the a_v4_extfn_blob structure to represent a free-standing blob object.

Implementation

```

typedef struct a_v4_extfn_blob {
    a_sql_uint64 (SQL_CALLBACK *blob_length) (a_v4_extfn_blob *blob);
    void (SQL_CALLBACK *open_istream) (a_v4_extfn_blob *blob,
a_v4_extfn_blob_istream **is);
    void (SQL_CALLBACK *close_istream) (a_v4_extfn_blob *blob,
a_v4_extfn_blob_istream *is);
    void (SQL_CALLBACK *release) (a_v4_extfn_blob *blob);
} a_v4_extfn_blob;

```

Method Summary

Method Name	Data Type	Description
blob_length	a_sql_uint64	Returns the length, in bytes, of the specified blob.
open_istream	void	Opens an input stream that can be used to begin reading from the specified blob.
close_istream	void	Closes the input stream for the specified blob.

Method Name	Data Type	Description
release	void	Indicates that the caller is done with this blob and that the blob owner is free to release resources. After release() , referencing the blob results in an error. The owner usually deletes the memory when release() is called.

Description

The object `a_v4_extfn_blob` is used when:

- a table UDF needs to read LOB or CLOB data from a scalar input value
- a TPF needs to read LOB or CLOB data from a column in an input table

Restrictions and Limitations

None.

blob_length

Use the `blob_length` v4 API method to return the length, in bytes, of the specified blob.

Declaration

```
a_sql_uint64 blob_length(
    a_v4_extfn_blob *
```

Usage

Returns the length, in bytes, of the specified blob.

Parameters

Parameter	Description
blob	The blob to return the length of.

Returns

The length of the specified blob.

See also

- *open_istream* on page 19
- *close_istream* on page 19

- *release* on page 20

open_istream

Use the `open_istream` v4 API method to open an input stream to read from a blob.

Declaration

```
void open_istream(
    a_v4_extfn_blob *blob,
    a_v4_extfn_blob_istream **is
)
```

Usage

Opens an input stream that can be used to begin reading from the specified blob.

Parameters

Parameter	Description
blob	The blob to open the input stream on.
is	An output parameter identifying the returned open input stream.

Returns

Nothing.

See also

- *blob_length* on page 18
- *close_istream* on page 19
- *release* on page 20

close_istream

Use the `close_istream` v4 API method to close the input stream for the specified blob.

Declaration

```
void close_istream(
    a_v4_extfn_blob *blob,
    a_v4_extfn_blob_istream *is
)
```

Usage

Closes the input stream previously opened with the `open_istream` API.

Parameters

Parameter	Description
blob	The blob to close the input stream on.
is	A parameter identifying the input stream to close.

Returns

Nothing.

See also

- *blob_length* on page 18
- *open_istream* on page 19
- *release* on page 20

release

Use the `release v4` API method to indicate that the caller is done with the currently selected blob. Releasing enables the owner to free memory.

Declaration

```
void release(
a_v4_extfn_blob *blob
)
```

Usage

Indicates that the caller is done with this blob and that the blob owner is free to release resources. After **release()**, referencing the blob results in an error. The owner usually deletes the memory when **release()** is called.

Parameters

Parameter	Description
blob	The blob to release.

Returns

Nothing.

See also

- *blob_length* on page 18
- *open_istream* on page 19
- *close_istream* on page 19

Blob Input Stream (a_v4_extfn_blob_istream)

Use the `a_v4_extfn_blob_istream` structure to read blob data for a LOB or CLOB scalar input column, or LOB or CLOB column in an input table.

Implementation

```
typedef struct a_v4_extfn_blob_istream {
    size_t (SQL_CALLBACK *get)( a_v4_extfn_blob_istream *is, void
*buf, size_t len );
    a_v4_extfn_blob      *blob;
    a_sql_byte           *beg;
    a_sql_byte           *ptr;
    a_sql_byte           *lim;
} a_v4_extfn_blob_istream;
```

Method Summary

Method Name	Data Type	Description
get	size_t	Gets a specified amount of data from a blob input stream.

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>Blob</i>	a_v4_extfn_blob	The underlying blob structure for which this input stream was created.
<i>Beg</i>	a_sql_byte	A pointer to the beginning of the current chunk of data.
<i>Ptr</i>	a_sql_byte	A pointer to the current byte in the chunk of data.
<i>Lim</i>	a_sql_byte	A pointer to the end of the current chunk of data.

get

Use the `get` v4 API method to get a specified amount of data from a blob input stream.

Declaration

```
size_t get(
    a_v4_extfn_blob_istream *is,
    void *buf,
    size_t len
```

)

Usage

Gets a specified amount of data from a blob input stream.

Parameters

Parameter	Description
is	The input stream to retrieve data from.
buf	The buffer to store the data in.
len	The amount of data to retrieve.

Returns

The amount of data received.

Column Data (a_v4_extfn_column_data)

The structure `a_v4_extfn_column_data` represents a single column's worth of data. This is used by the producer when generating result set data, or by the consumer when reading input table column data.

Implementation

```
typedef struct a_v4_extfn_column_data {
    a_sql_byte      *is_null;
    a_sql_byte      null_mask;
    a_sql_byte      null_value;

    void            *data;
    a_sql_uint32    *piece_len;
    size_t          max_piece_len;

    void            *blob_handle;
} a_v4_extfn_column_data;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>is_null</i>	a_sql_byte *	Points to a byte where the NULL information for the value is stored.
<i>null_mask</i>	a_sql_byte	One or more bits used to represent the NULL value
<i>null_value</i>	a_sql_byte	The value representing NULL

Data Member	Data Type	Description
<i>data</i>	void *	Pointer to the data for the column. Depending on the type of fetch mechanism, either points to an address in the consumer, or an address where the data is stored in the UDF.
<i>piece_len</i>	a_sql_uint32 *	The actual length of data for variable-length data types
<i>max_piece_len</i>	size_t	The maximum data length allowed for this column.
<i>blob_handle</i>	void *	A non-NULL value means that the data for this column must be read using the <code>blob</code> API

Description

The `a_v4_extfn_column_data` structure represents the data values and related attributes for a specific data column. This structure is used by the producer when generating result set data. Data producers are also expected to create storage for *data*, *piece_len*, and the *is_null* flag.

The *is_null*, *null_mask*, and *null_value* data members indicate null in a column, and handle situations in which the null-bits are encoded into one byte for eight columns, or other cases in which a full byte is used for each column.

This example shows how to interpret the three fields used to represent NULL: *is_null*, *null_mask*, and *null_value*.

```
is_value_null()
    return( (*is_null & null_mask) == null_value )

set_value_null()
    *is_null = ( *is_null & ~null_mask) | null_value

set_value_not_null()
    *is_null = *is_null & ~null_mask | (~null_value & null_mask)
```

Column List (a_v4_extfn_column_list)

Use the `a_v4_extfn_column_list` structure to provide a list of columns when describing **PARTITION BY** or to provide a list of columns when describing **TABLE_UNUSED_COLUMNS**.

Implementation

```
typedef struct a_v4_extfn_column_list {
    a_sql_int32      number_of_columns;
    a_sql_uint32    column_indexes[1];    // there are
number_of_columns entries
} a_v4_extfn_column_list;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>number_of_columns</i>	a_sql_uint32	The number of columns in the list.
<i>column_indexes</i>	a_sql_uint32 *	A contiguous array of size <i>number_of_columns</i> with the column indexes (1-based).

Description

The meaning of the contents of the column list changes, depending on whether the list is used with **TABLE_PARTITIONBY** or **TABLE_UNUSED_COLUMNS**.

Column Order (a_v4_extfn_order_el)

Use the `a_v4_extfn_order_el` structure to describe the element order in a column.

Implementation

```
typedef struct a_v4_extfn_order_el {
    a_sql_uint32    column_index;    // Index of the column in the
table (1-based)
    a_sql_byte     ascending;        // Nonzero if the column
is ordered "ascending".
} a_v4_extfn_order_el;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>column_index</i>	a_sql_uint32	Index of the column in the table (1-based).
<i>ascending</i>	a_sql_byte	Nonzero, if the column order is "ascending."

Description

The `a_v4_extfn_order_el` structure describes a column and tells whether it should be in ascending or descending order. The `a_v4_extfn_orderby_list` structure holds an array of these structures. There is one `a_v4_extfn_order_el` structure for each column in the **ORDERBY** clause.

Column Subset (a_v4_extfn_col_subset_of_input)

Use the `a_v4_extfn_col_subset_of_input` structure to declare that an output column has a value that is always taken from a particular input column to the UDF.

Implementation

```
typedef struct a_v4_extfn_col_subset_of_input {
    a_sql_uint32    source_table_parameter_arg_num;    // arg_num of
the source table parameter
    a_sql_uint32    source_column_number;            // source column of
the source table
} a_v4_extfn_col_subset_of_input;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<code>source_table_parameter_arg_num</code>	<code>a_sql_uint32 *</code>	<code>arg_num</code> of the source TABLE parameter
<code>source_column_number</code>	<code>a_sql_uint32 *</code>	Source column of the source table

Description

The query optimizer uses the subset of input to infer logical properties of the values in the output column. For example, the number of distinct values in the input column is an upper bound on the distinct values in the output column, and any local predicates on the input column also hold on the output column.

Describe API

The `_describe_extfn` function is a member of `a_v4_extfn_proc`. A UDF gets and sets logical properties using the `describe_column`, `describe_parameter`, and `describe_udf` properties in the `a_v4_extfn_proc_context` object.

_describe_extfn Declaration

```
void (UDF_CALLBACK *_describe_extfn)(a_v4_extfn_proc_context
*cntxt );
)
```

Usage

The `_describe_extfn` function describes the procedure evaluation to the server.

Each of the `describe_column`, `describe_parameter`, and `describe_udf` properties has an associated get and set method, a set of attribute types, and an associated data type for each attribute. The get methods retrieve information from the server; the set methods describe the logical properties of the UDF (such as the number of output columns or the number of distinct values for a output column) to the server.

***describe_column_get**

The `describe_column_get` v4 API method is used by the table UDF to retrieve properties about an individual column of a TABLE parameter.

Declaration

```
a_sql_int32 (SQL_CALLBACK *describe_column_get) (
    a_v4_extfn_proc_context    *cntxt,
    a_sql_uint32               arg_num,
    a_sql_uint32               column_num,
    a_v4_extfn_describe_parm_type describe_type,
    void                       *describe_buffer,
    size_t                     describe_buffer_len );
```

Parameters

Parameter	Description
cntxt	The procedure context object for this UDF.
arg_num	The ordinal of the TABLE parameter (0 is the result table, 1 for first input argument).
column_num	The ordinal of the column starting at 1.
describe_type	A selector indicating what property to retrieve.
describe_buffer	A structure that holds the describe information for the specified property to get from the server. The specific structure or data type is indicated by the describe_type parameter.
describe_buffer_length	The length, in bytes, of the describe_buffer .

Returns

On success, returns the number of bytes written to the **describe_buffer**. If an error occurs, or no property is retrieved, this function returns one of the generic `describe_column` errors.

*Attributes for *describe_column_get*

Code showing the attributes for `describe_column_get` v4 API method.

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
```

```
EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
} a_v4_extfn_describe_col_type;
```

EXTFNAPIV4_DESCRIBE_COL_NAME (Get)

The **EXTFNAPIV4_DESCRIBE_COL_NAME** attribute indicates the column name. Used in a `describe_column_get` scenario.

Data Type

char[]

Description

The column name. This property is valid only for table arguments.

Usage

If a UDF gets this property, then the name of the specified column is returned.

Returns

On success, returns the length of the column name.

On failure, returns one of the generic `describe_column` errors, or:

- **EXTFNAPIV4_DESCRIBE_INVALID_STATE** – get error returned if the query processing phase is not greater than `Initial`.
- **EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH** – get error returned if the buffer length has insufficient characters or is 0 length.
- **EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER** – get error returned if the parameter is not a `TABLE` parameter.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_COL_TYPE (Get)

The **EXTFNAPIV4_DESCRIBE_COL_TYPE** attribute indicates the data type of the column. Used in a `describe_column_get` scenario.

Data Type

a_sql_data_type

Description

The data type of the column. This property is valid only for table arguments.

Usage

If a UDF gets this property, then returns the data type of the specified column.

Returns

On success, the `sizeof(a_sql_data_type)` is returned.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_sql_data_type`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than `Initial`.

Query Processing Phases

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_COL_WIDTH (Get)

The `EXTFNAPIV4_DESCRIBE_COL_WIDTH` attribute indicates the width of the column.

Used in a `describe_column_get` scenario.

Data Type

`a_sql_uint32`

Description

The width of a column. Column width is the amount of storage, in bytes, required to store a value of the associated data type. This property is valid only for table arguments.

Usage

If a UDF gets this property, then returns the width of the column as defined in the **CREATE PROCEDURE** statement.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than Initial.

Query Processing Phases

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_COL_SCALE (Get)

The `EXTFNAPIV4_DESCRIBE_COL_SCALE` attribute indicates the scale of the column. Used in a `describe_column_get` scenario.

Data Type

`a_sql_uint32`

Description

The scale of a column. For arithmetic data types, parameter scale is the number of digits to the right of the decimal point in a number. This property is valid only for table arguments.

Usage

If the UDF gets this property, returns the scale of the column as defined in the **CREATE PROCEDURE** statement. This property is valid only for arithmetic data types.

Returns

On success, returns the `sizeof(a_sql_uint32)` if the value was returned, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – get error returned if the scale is unavailable for the data type of the specified column.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than Initial.

Query Processing Phases

- Annotation phase
- Query Optimization phase
- Plan Building phase

- Execution phase

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL (Get)

The **EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL** attribute indicates if the column can be NULL. Used in a `describe_column_get` scenario.

Data Type

`a_sql_byte`

Description

True, if the column can be NULL. This property is valid only for table arguments. This property is valid only for argument 0.

Usage

If a UDF gets this property, returns 1 if the column can be NULL, and returns 0 if otherwise.

Returns

On success, returns the `sizeof(a_sql_byte)` if the attribute is available, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – returned if the attribute was not available to get. This can happen if the column was not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the specified argument is an input table and the query processing phase is not greater than Plan Building phase.

Query Processing Phases

Valid in:

- Execution phase

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES (Get)

The **EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES** attribute describes the distinct values for a column. Used in a `describe_column_get` scenario.

Data Type

`a_v4_extfn_estimate`

Description

The estimated number of distinct values for a column. This property is valid only for table arguments.

Usage

If a UDF gets this property, it returns the estimated number of distinct values for a column.

Returns

On success, returns the `sizeof(a_v4_extfn_estimate)`, if it returns a value, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – returned if the attribute was not available to get. This can happen if the column was not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_v4_extfn_estimate`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the specified argument is an input table and the query processing phase is greater than Optimization.

Query Processing Phases

Valid in:

- Plan Building phase
- Execution phase

Example

Consider this procedure definition and code fragment in the `_describe_extfn` API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
RESULTS ( r1 INT, r2 INT, r3 INT )
EXTERNAL 'my_tpf_proc@mylibrary';
```

```
CREATE TABLE T( x INT, y INT, z INT );
```

```
select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

This example shows how a TPF gets the number of distinct values for column one of the input table. A TPF may want to get this value, if it is beneficial for choosing an appropriate processing algorithm.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_PLAN_BUILDING ) {
        a_v4_extfn_estimate num_distinct;

        a_sql_int32 ret = 0;

        // Get the number of distinct values expected from the first
column
        // of the table input parameter 'col_table'
        ret = cntxt->describe_column_get( cntxt, 2, 1
EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
```

```
        &num_distinct,  
        sizeof(a_v4_extfn_estimate) );  
  
    // default algorithm is 1  
    _algorithm = 1;  
  
    if( ret > 0 ) {  
        // choose the best algorithm for sample size.  
  
        if ( num_distinct.value < 100 ) {  
            // use faster algorithm for small distinct values.  
            _algorithm = 2;  
        }  
    }  
    else {  
        if ( ret < 0 ) {  
            // Handle the error  
            //   or continue with default algorithm  
        } else {  
            // Attribute was unavailable  
            // We will use the default algorithm.  
        }  
    }  
}  
}
```

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE (Get)

The **EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE** attribute indicates if a column is unique in the table. Used in a `describe_column_get` scenario.

Data Type

`a_sql_byte`

Description

True, if the column is unique within the table. This property is valid only for table arguments.

Usage

If the UDF gets this property, then returns 1 if the column is unique, and 0 otherwise.

Returns

On success, returns the `sizeof(a_sql_byte)` or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – if the attribute was unavailable to get. This can happen if the column was not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_sql_byte`.

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than Initial.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT (Get)

The `EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT` attribute indicates if a column is constant. Used in a `describe_column_get` scenario.

Data Type

`a_sql_byte`

Description

True, if the column is constant for the lifetime of the statement. This property is valid only for input table arguments.

Usage

If a UDF gets this property, the return value is 1 if the column is constant for the lifetime of the statement and 0 otherwise. Input table columns are constant, if the column in the select list for the input table is a constant expression or NULL.

Returns

On success, returns the `sizeof(a_sql_byte)`, if the value was returned, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – the attribute is not available to get. Returned, if the column is not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned, if the describe buffer is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned, if the query processing phase is not greater than Initial.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned, if the specified argument is not an input table.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE (Get)

The **EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE** attribute indicates the constant value of a column. Used in a `describe_column_get` scenario.

Data Type

`an_extfn_value`

Description

The value of the column, if it is constant for the statement lifetime. If `EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT` for this column returns true, this value is available. This property is valid only for table arguments.

Usage

For columns of input tables that have a constant value, the value is returned. If the value is unavailable, then NULL is returned.

Returns

On success, returns the `sizeof(a_sql_byte)`, if the value was returned, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – the attribute is not available to get. Returned, if the column is not involved in the query, or if the value is not considered constant.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned, if the describe buffer is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned, if the query processing phase is not greater than Initial.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned, if the specified argument is not an input table.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER (Get)

The **EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER** attribute indicates if a column in the result table is used by the consumer. Used in a `describe_column_get` scenario.

Data Type

`a_sql_byte`

Description

Used either to determine whether a column in the result table is used by the consumer, or to indicate that a column in an input is not needed. Valid for table arguments. Allows the user to set or retrieve information about a single column, whereas the similar attribute `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` sets or retrieves information about all columns in a single call.

Usage

The UDF queries this property to determine if a result table column is required by the consumer. This can help the UDF avoid unnecessary work for unused columns.

Returns

On success, returns the `sizeof(a_sql_byte)` or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – if the attribute was unavailable to get. This can happen if the column was not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_v4_extfn_estimate`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than `Initial`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the argument specified is not argument 0.

Query Processing Phases

Valid during:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

The **PROCEDURE** definition and code fragment in the `_describe_extfn` API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2
```

```
INT ) )
  RESULTS ( r1 INT, r2 INT, r3 INT )
  EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select r2,r3 from my_tpf( 'test', TABLE( select x,y from T ) )
```

When this TPF runs, it is beneficial to know if the user has selected column `r1` of the result set. If the user does not need `r1`, calculations for `r1` may be unnecessary and we do not need to produce it for the server.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
        a_sql_byte col_is_used = 0;
        a_sql_int32 ret = 0;

        ret = cntxt->describe_column_get( cntxt, 0, 1,
            EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
            &col_is_used,
            sizeof(a_sql_byte) );

        if( ret < 0 ) {
            // Handle the error.
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE (Get)

The **EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE** attribute indicates the minimum value for a column. Used in a `describe_column_get` scenario.

Data Type

`an_extfn_value`

Description

The minimum value for a column, if available. Valid only for argument 0 and table arguments.

Usage

If a UDF gets the **EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE** property, the minimum value of the column data is returned in the `describe_buffer`. If the input table is a base table, the minimum value is based on all of the column data in the table and is accessible only if there is an index on the table column. If the input table is the result of another UDF, the minimum value is the `EXTFNAPIV4_DESCRIBE_COL_TYPE` set by that UDF.

The data type for this property is different for different columns. The UDF can use `EXTFNAPIV4_DESCRIBE_COL_TYPE` to determine the data type of the column. The UDF

can also use `EXTFNAPIV4_DESCRIBE_COL_WIDTH` to determine the storage requirements of the column, to provide an equivalently sized buffer to hold the value.

`describe_buffer_length` allows the server to determine if the buffer is valid.

If the `EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE` property is unavailable, `describe_buffer` is `NULL`.

Returns

On success, returns the `describe_buffer_length`, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – if the attribute was unavailable to get. Returned if the column was not involved in the query or the minimum value was unavailable for the requested column.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – Get error returned, if the describe buffer is not large enough to hold the minimum value.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – Get error returned if the state is not greater than Initial.

Query Processing States

Valid in any state except Initial state:

- Annotation state
- Query Optimization state
- Plan Building state
- Execution state

Example

The procedure definition and code fragment in the `_describe_extfn` API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
  RESULTS ( r1 INT, r2 INT, r3 INT )
  EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

This example illustrates how a TPF would get the minimum value for column two of the input table, for internal optimization purposes.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
        a_sql_int32 min_value = 0;
        a_sql_int32 ret = 0;
```

```
// Get the minimum value of the second column of the
// table input parameter 'col_table'

ret = cntxt->describe_column_get( cntxt, 2, 2
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    &min_value,
    sizeof(a_sql_int32) );

if( ret < 0 ) {
    // Handle the error.
}

}
```

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE (Get)

The **EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE** attribute indicates the maximum value for the column. Used in a `describe_column_get` scenario.

Data Type

`an_extfn_value`

Description

The maximum value for a column. This property is valid only for argument 0 and table arguments.

Usage

If a UDF gets the **EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE** property, then the maximum value of the column data is returned in the **describe_buffer**. If the input table is a base table, the maximum value is based on all of the column data in the table and is accessible only if there is an index on the table column. If the input table is the result of another UDF, the maximum value is the **COL_MAXIMUM_VALUE** set by that UDF.

The data type for this property is different for different columns. The UDF can use **EXTFNAPIV4_DESCRIBE_COL_TYPE** to determine the data type of the column. The UDF can also use **EXTFNAPIV4_DESCRIBE_COL_WIDTH** to determine the storage requirements of the column, to provide an equivalently sized buffer to hold the value.

describe_buffer_length allows the server to determine if the buffer is valid.

If **EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE** is unavailable, **describe_buffer** is NULL.

Returns

On success, returns the `describe_buffer_length` or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – If the attribute was unavailable to get. This can happen if the column was uninvolved in the query, or if the maximum value was unavailable for the requested column.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – Get error returned if the describe buffer is not large enough to hold the maximum value.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – Get error returned if the query processing phase is not greater than Initial.

Query Processing Phases

Valid in any phase except Initial phase:

- Annotation phase
- Query Optimization phase
- Plan building phase
- Execution phase

Example

The **PROCEDURE** definition and code fragment in the `_describe_extfn` API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

This example illustrates how a TPF would get the maximum value for column two of the input table, for internal optimization purposes.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
        a_sql_int32 max_value = 0;
        a_sql_int32 ret = 0;

        // Get the maximum value of the second column of the
        // table input parameter 'col_table'
        ret = cntxt->describe_column_get( cntxt, 2, 2
            EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
            &max_value,
            sizeof(a_sql_int32) );

        if( ret < 0 ) {
            // Handle the error.
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT (Get)

The **EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT** attribute sets a subset of the values specified in an input column. Using this attribute in a `describe_column_get` scenario returns an error.

Data Type

`a_v4_extfn_col_subset_of_input`

Description

Column values are a subset of the values specified in an input column.

Usage

This attribute can be set only.

Returns

Returns the error `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE`.

Query Processing States

Error `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` is returned in any state.

***describe_column_set**

The `describe_column_set` v4 API method sets UDF column-level properties on the server.

Description

Column-level properties describe various characteristics about columns in the result set or input tables in a TPF. For example, a UDF can tell the server that a column in its result set will have only ten distinct values.

Declaration

```
a_sql_int32 (SQL_CALLBACK *describe_column_set) (
    a_v4_extfn_proc_context      *cntxt,
    a_sql_uint32                 arg_num,
    a_sql_uint32                 column_num,
    a_v4_extfn_describe_udf_type describe_type,
    const void                   *describe_buffer,
    size_t                       describe_buffer_len );
```

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object for this UDF.

Parameter	Description
arg_num	The ordinal of the TABLE parameter (0 is the result table, 1 for first input argument).
column_num	The ordinal of the column starting at 1.
describe_type	A selector indicating what property to set.
describe_buffer	A structure that holds the describe information for the specified property to set on the server. The specific structure or data type is indicated by the describe_type parameter.
describe_buffer_length	The length, in bytes, of the describe_buffer .

Returns

On success, returns the number of bytes written to the **describe_buffer**. If an error occurs, or no property is retrieved, this function returns one of the generic `describe_column` errors.

Attributes for `*describe_column_set`

Code showing the attributes for `describe_column_set`.

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
} a_v4_extfn_describe_col_type;
```

`EXTFNAPIV4_DESCRIBE_COL_NAME` (Set)

The `EXTFNAPIV4_DESCRIBE_COL_NAME` attribute indicates a column name. Used in a `describe_column_set` scenario.

Data Type

`char[]`

Description

The column name. This property is valid only for table arguments.

Usage

For argument 0, if the UDF sets this property, the server compares the value with the name of the column supplied in the **CREATE PROCEDURE** statement. The comparison ensures that the **CREATE PROCEDURE** statement has the same column name as expected by the UDF.

Returns

On success, returns the length of the column name.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not Annotation.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – set error returned if the parameter is not a `TABLE` parameter.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the length of input column name exceeds 128 characters or if the input column name and column name stored in the catalog do not match.

Query Processing States

- Annotation state

Example

```
short desc_rc = 0;
char name[7] = 'column1';
// Verify that the procedure was created with the second column
of the result table as an int
if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_NAME,
                                     name,
                                     sizeof(name) );

    if( desc_rc < 0 ) {
        // handle the error.
    }
}
```

EXTFNAPIV4_DESCRIBE_COL_TYPE (Set)

The `EXTFNAPIV4_DESCRIBE_COL_TYPE` attribute indicates the data type of the column. Used in a `describe_column_set` scenario.

Data Type

`a_sql_data_type`

Description

The data type of the column. This property is valid only for table arguments.

Usage

For argument zero, if the UDF sets this property, then the server compares the value with the data type of the column supplied in the **CREATE PROCEDURE** statement. This allows the UDF to ensure the **CREATE PROCEDURE** statement has the same data type as expected by the UDF.

Returns

On success, returns the `a_sql_data_type`.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – Set error returned if the describe buffer is not the size of `a_sql_data_type`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – Set error returned if the state is not Annotation.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – Set error returned if the input data type and the data type stored in the catalog do not match,.

Query processing states

- Annotation state

Example

```
short desc_rc = 0;
a_sql_data_type type = DT_INT;

    // Verify that the procedure was created with the second column of
    // the result table as an int
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_TYPE,
        &type,
            sizeof(a_sql_data_type) );
        if( desc_rc < 0 ) {
            // handle the error.
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_COL_WIDTH (Set)

The `EXTFNAPIV4_DESCRIBE_COL_WIDTH` attribute indicates the width of the column. Used in a `describe_column_set` scenario.

Data Type

`a_sql_uint32`

Description

The width of a column. Column width is the amount of storage, in bytes, required to store a value of the associated data type. This property is valid only for table arguments.

Usage

If the UDF sets this property, the server compares the value with the width of the column supplied in the **CREATE PROCEDURE** statement. This allows the UDF to ensure the **CREATE PROCEDURE** statement has the same column width as expected by the UDF.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the describe buffer is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the query processing state is not Annotation.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the input width and width stored in the catalog do not match.

Query Processing States

Valid in:

- Annotation state

EXTFNAPIV4_DESCRIBE_COL_SCALE (Set)

The `EXTFNAPIV4_DESCRIBE_COL_SCALE` attribute indicates the scale of the column. Used in a `describe_column_set` scenario.

Data Type

`a_sql_uint32`

Description

The scale of a column. For arithmetic data types, parameter scale is the number of digits to the right of the decimal point in a number. This property is valid only for table arguments.

Usage

If the UDF sets this property, the server compares the value with the scale of the column supplied in the **CREATE PROCEDURE** statement. This allows the UDF to ensure the **CREATE PROCEDURE** statement has the same column width as expected by the UDF. This property is valid only for arithmetic data types.

Returns

On success, returns the `sizeof(a_sql_uint32)`, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – set error returned if the scale is not available for the data type of the specified column.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the describe buffer is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the query processing state is not Annotation.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the input scale and scale stored in the catalog do not match.

Query Processing States

Valid in:

- Annotation state

Example

```

short desc_rc = 0;
a_sql_uint32 scale = 0;

    // Verify that the procedure has a scale of zero for the
second result table column.
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_SCALE,
        &scale,
        sizeof(a_sql_data_type) );
        if( desc_rc < 0 ) {
            // handle the error.
        }
    }

```

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL (Set)

The `EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL` attribute indicates if the column can be null. Used in a `describe_column_set` scenario.

Data Type

`a_sql_byte`

Description

True, if the column can be NULL. This property is valid only for table arguments. This property is valid only for argument 0.

Usage

The UDF can set this property for a result table column if that column can be NULL. If the UDF does not explicitly set this property, it is assumed that the column can be NULL. The server can use this information during the Optimization state.

Returns

On success, returns the `sizeof(a_sql_byte)` if the attribute was set or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – returned if the attribute was unavailable to set, which may happen if the column was uninvolved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the describe buffer is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not equal to `OPTIMIZATION`.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF attempts to set this attribute to a value other than 0 or 1.

Query Processing States

Valid in:

- Optimization state

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES (Set)

The `EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES` attribute describes the distinct values for a column. Used in a `describe_column_set` scenario.

Data Type

`a_v4_extfn_estimate`

Description

The estimated number of distinct values for a column. This property is valid only for table arguments.

Usage

The UDF can set this property if it knows how many distinct values a column can have in its result table. The server uses this information during the Optimization state.

Returns

On success, returns the `sizeof(a_v4_extfn_estimate)`, if it sets the value, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – returned if the attribute was unavailable to set. This can happen if the column was not involved in the query.

On failure, returns:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the describe buffer is not the size of `a_v4_extfn_estimate`.

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not equal to Optimization.

Query Processing States

Valid in:

- Optimization state

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE (Set)

The `EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE` attribute indicates if the column is unique in the table. Used in a `describe_column_set` scenario.

Data Type

`a_sql_byte`

Description

True, if the column is unique within the table. This property is valid only for table arguments.

Usage

The UDF can set this property if it knows the result table column value is unique. The server uses this information during the Optimization state. The UDF can set this property only for argument 0.

Returns

On success, returns the `sizeof(a_sql_byte)` or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – if the attribute was not available to set. This can happen if the column was not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the describe buffer is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the query processing state is not Optimization.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the `arg_num` is not zero.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF attempts to set this attribute to a value other than 0 or 1.

Query Processing States

Valid in:

- Optimization state

EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT (Set)

The **EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT** attribute indicates if the column is constant. Used in a `describe_column_set` scenario.

Data Type

`a_sql_byte`

Description

True, if the column is constant for the lifetime of the statement. This property is valid only for input table arguments.

Usage

This is a read only property. All attempts to set it return `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE`.

Returns

- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` – this is a read-only property; all attempts to set return this error.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned, if the state is not Optimization.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned, if the **arg_num** is not zero.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned, if the UDF attempts to set this attribute to a value other than 0 or 1.

Query Processing States

Not applicable.

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE (Set)

The **EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE** attribute indicates the constant value of the column. Used in a `describe_column_set` scenario.

Data Type

`an_extfn_value`

Description

The value of the column, if it is constant for the statement lifetime. If `EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT` for this column returns true, this value is available. This property is valid only for table arguments.

Usage

This property is read-only.

Returns

- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` – this is a read-only property; all attempts to set return this error.

Query Processing States

Not applicable.

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER (Set)

The `EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER` attribute indicates if the column in the result table is used by the consumer. Used in a `describe_column_set` scenario.

Data Type

`a_sql_byte`

Description

Used either to determine whether a column in the result table is used by the consumer, or to indicate that a column in an input is not needed. Valid for table arguments. Allows the user to set or retrieve information about a single column, whereas the similar attribute `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` sets or retrieves information about all columns in a single call.

Usage

The UDF sets `EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER` on columns in an input table to inform the producer that it does not need values for the column.

Returns

On success, returns the `sizeof(a_sql_byte)` or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – if the attribute was not available to set. This can happen if the column was not involved in the query.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the `describe` buffer is not the size of `a_v4_extfn_estimate`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the argument specified is argument 0.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not equal to `Optimization`.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the value the UDF is setting is not 0 or 1.

Query Processing States

Valid during:

- Optimization state

The PROCEDURE definition and code fragment in the `_describe_extfn` API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2
INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select r2,r3 from my_tpf( 'test', TABLE( select x,y from T ) )
```

When this TPF runs, it is beneficial for the server to know if column `y` is used by this TPF. If the TPF does not need `y`, the server can use this knowledge for optimization and does not send this column information to the TPF.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_byte col_is_used = 0;
        a_sql_int32 ret = 0;

        ret = cntxt->describe_column_get( cntxt, 2, 2,
            EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
            &col_is_used,
            sizeof(a_sql_byte) );

        if( ret < 0 ) {
            // Handle the error.
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE (Set)

The **EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE** attribute indicates the minimum value for the column. Used in a `describe_column_set` scenario.

Data Type

`an_extfn_value`

Description

The minimum value a column can have, if available. Only valid for argument 0.

Usage

The UDF can set `EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE`, if it knows what the minimum data value of the column is. The server can use this information during optimization.

The UDF can use `EXTFNAPIV4_DESCRIBE_COL_TYPE` to determine the data type of the column, and `EXTFNAPIV4_DESCRIBE_COL_WIDTH` to determine the storage requirements of the column, to provide an equivalently sized buffer to hold the value to set.

Returns

On success, returns the `describe_buffer_length`, or:

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – if the attribute cannot be set. Returned if the column was not involved in the query or the minimum value was not available for the requested column.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned, if the describe buffer is not large enough to hold the minimum value.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned, if the state is not equal to `Optimization`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned, if the `arg_num` is not 0.

Query Processing States

Valid in:

- Optimization state

Example

The **PROCEDURE** definition and UDF code fragment that implements the `_describe_extfn` callback API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
  RESULTS ( r1 INT, r2 INT, r3 INT )
  EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

This example shows a TPF where it is useful to the server (or to another TPF that takes the result of this TPF as input) to know the minimum value of result set column one. In this instance, the minimum output value of column one is 27.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
```

```
if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_int32 min_value = 27;
    a_sql_int32 ret = 0;

// Tell the server what the minimum value of the first column
// of our result set will be.

    ret = cntxt->describe_column_set( cntxt, 0, 1
        EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
        &min_value,
        sizeof(a_sql_int32) );

    if( ret < 0 ) {
        // Handle the error.
    }
}
}
```

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE (Set)

The **EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE** attribute indicates the maximum value for the column. Used in a `describe_column_set` scenario.

Data Type

`an_extfn_value`

Description

The maximum value for a column. This property is valid only for argument 0 and table arguments.

Usage

The UDF can set **EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE**, if it knows what the maximum data value of the column is. The server can use this information during optimization.

The UDF can use **EXTFNAPIV4_DESCRIBE_COL_TYPE** to determine the data type of the column, and **EXTFNAPIV4_DESCRIBE_COL_WIDTH** to determine the storage requirements of the column, to provide an equivalently sized buffer to hold the value to set.

`describe_buffer_length` is the `sizeof()` this buffer.

Returns

On success, returns the `describe_buffer_length`, if the value was set, or:

- **EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE** – if the attribute could not be set. Returned if the column was not involved in the query or the maximum value was not available for the requested column.

On failure, returns one of the generic `describe_column` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned, if the describe buffer is not large enough to hold the maximum value.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – Set error returned, if the query processing state is not equal to Optimization.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned, if the `arg_num` is not 0.

Query Processing States

Valid in:

- Optimization state

Example

The PROCEDURE definition and UDF code fragment that implements the `_describe_extfn` callback API function:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

This example shows a TPF where it is useful to the server (or to another TPF that takes the result of this TPF as input) to know the maximum value of result set column one. In this instance, the maximum output value of column one is 500000.

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_int32 max_value = 500000;
        a_sql_int32 ret = 0;

        // Tell the server what the maximum value of the first column
        // of our result set will be.

        ret = cntxt->describe_column_set( cntxt, 0, 1
            EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
            &max_value,
            sizeof(a_sql_int32) );

        if( ret < 0 ) {
            // Handle the error.
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT (Set)

The **EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT** attribute sets a subset of the values specified in an input column. Used in a `describe_column_set` scenario.

Data Type

`a_v4_extfn_col_subset_of_input`

Description

Column values are a subset of the values specified in an input column.

Usage

Setting this describe attribute informs the query optimizer that the indicated column values are a subset of those values specified in an input column. For example, consider a filter TPF that consumes a table and filters out rows based on a function. In such a case, the return table is a subset of the input table. Setting

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT for the filter TPF optimizes the query.

Returns

On success, returns the `sizeof(a_v4_extfn_col_subset_of_input)`.

On failure, returns one of the generic `describe_column` errors, or:

- **EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH** – set error returned if the buffer length is less than `sizeof(a_v4_extfn_col_subset_of_input)`.
- **EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE** – set error returned if the column index of the source table is out of range.
- **EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE** – set error returned if the column `subset_of_input` is set on is not applicable (for example, if the column is not in the select list).
- **EXTFNAPIV4_DESCRIBE_INVALID_STATE** – set error returned if the query processing state is not **Optimization**.
- **EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH** – set error returned if the buffer length is zero.
- **EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER** – set error returned if called on a parameter other than the return table.

Query Processing States

Valid in:

- Optimization state

Example

```
a_v4_extfn_col_subset_of_input colMap;
```



```
colMap.source_table_parameter_arg_num = 4;
colMap.source_column_number = i;

desc_rc = ctx->describe_column_set( ctx,
    0, i,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    &colMap, sizeof(a_v4_extfn_col_subset_of_input) );
```

***describe_parameter_get**

The `describe_parameter_get` v4 API method gets UDF parameter properties from the server.

Declaration

```
a_sql_int32 (SQL_CALLBACK *describe_parameter_get) (
    a_v4_extfn_proc_context      *cntxt,
    a_sql_uint32                 arg_num,
    a_v4_extfn_describe_udf_type describe_type,
    const void                   *describe_buffer,
    size_t                       describe_buffer_len );
```

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object.
<code>arg_num</code>	The ordinal of the TABLE parameter (0 is for the result table and 1 is for first input argument)
<code>describe_type</code>	A selector indicating what property to set.
<code>describe_buffer</code>	A structure that holds the describe information for the specified property to set on the server. The specific structure or data type is indicated by the describe_type parameter.
<code>describe_buffer_length</code>	The length, in bytes, of the describe_buffer .

Returns

On success, returns 0 or the number of bytes written to the **describe_buffer**. A value of 0 indicates that the server was unable to get the attribute, but no error condition occurred. If an error occurred, or no property was retrieved, this function returns one of the generic **describe_parameter** errors.

*Attributes for *describe_parameter_get*

Code showing the attributes for `describe_parameter_get`.

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
```

```
EXTFNAPIV4_DESCRIBE_PARM_TYPE,  
EXTFNAPIV4_DESCRIBE_PARM_WIDTH,  
EXTFNAPIV4_DESCRIBE_PARM_SCALE,  
EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,  
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,  
EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,  
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,  
  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,  
EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,  
  
} a_v4_extfn_describe_parm_type;
```

EXTFNAPIV4_DESCRIBE_PARM_NAME Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_NAME` attribute indicates the parameter name. Used in a `describe_parameter_get` scenario.

Data Type

char[]

Description

The name of a parameter to a UDF.

Usage

Gets the parameter name as defined in the **CREATE PROCEDURE** statement. Invalid for parameter 0.

Returns

On success, returns the length of the parameter name.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not large enough to hold the name.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than **Initial**.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the parameter is the result table.

Query Processing Phases

Valid in:

- Annotation phase

- Query optimization phase
- Plan building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_PARM_TYPE Attribute (Get)

The **EXTFNAPIV4_DESCRIBE_PARM_TYPE** attribute returns the data type in a `describe_parameter_get` scenario.

Data Type

`a_sql_data_type`

Description

The data type of a parameter to a UDF.

Usage

Gets the data type of the parameter as defined in the **CREATE PROCEDURE** statement.

Returns

On success, returns `sizeof(a_sql_data_type)`.

On failure, returns one of the generic `describe_parameter` errors or:

- **EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH** – get error returned if the **describe_buffer** is not the `sizeof(a_sql_data_type)`.
- **EXTFNAPIV4_DESCRIBE_INVALID_STATE** – get error returned if the query processing phase is not greater than **Initial**.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_PARM_WIDTH Attribute (Get)

The **EXTFNAPIV4_DESCRIBE_PARM_WIDTH** attribute indicates the width of a parameter. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_uint32`

Description

The width of a parameter to a UDF. `EXTFNAPIV4_DESCRIBE_PARAM_WIDTH` applies only to scalar parameters. Parameter width is the amount of storage, in bytes, required to store a parameter of the associated data type.

- **Fixed length data types** – the bytes required to store the data.
- **Variable length data types** – the maximum length.
- **LOB data types** – the amount of storage required to store a handle to the data.
- **TIME data types** – the amount of storage required to store the encoded time.

Usage

Gets the width of the parameter as defined in the **CREATE PROCEDURE** statement.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than `Initial`.
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the specified parameter is a `TABLE` parameter. This includes parameter 0, or parameter *n* where *n* is an input table.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

Example

Sample procedure definition:

```
CREATE PROCEDURE my_udf(IN p1 INT, IN p2 char(100))
RESULT (x INT)
EXTERNAL NAME 'my_udf@myudflib';
```

Sample `_describe_extfn` API function code fragment:

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_uint32 width = 0;
```

```

a_sql_int32 ret = 0;

// Get the width of parameter 1
ret = cntxt->describe_parameter_get( cntxt, 1,
EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
&width,
sizeof(a_sql_uint32) );

if( ret < 0 ) {
    // Handle the error.
}

//Allocate some storage based on parameter width
a_sql_byte *p = (a_sql_byte *)cntxt->alloc( cntxt, width )

// Get the width of parameter 2
ret = cntxt->describe_parameter_get( cntxt, 2,
EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
&width,
sizeof(a_sql_uint32) );
if( ret <= 0 ) {
    // Handle the error.
}

// Allocate some storage based on parameter width
char *c = (char *)cntxt->alloc( cntxt, width )

...
}
}

```

EXTFNAPIV4_DESCRIBE_PARM_SCALE Attribute (Get)

The **EXTFNAPIV4_DESCRIBE_PARM_SCALE** attribute indicates the scale of a parameter. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_uint32`

Description

The scale of a parameter to a UDF. For arithmetic data types, parameter scale is the number of digits to the right of the decimal point in a number.

This attribute is not valid for:

- non-arithmetic data types
- TABLE parameters

Usage

Gets the scale of the parameter as defined in the **CREATE PROCEDURE** statement.

Returns

On success, returns the size of (a_sql_uint32).

On failure, returns one of the generic describe_parameter errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than `Initial`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the specified parameter is a `TABLE` parameter. This includes parameter 0, or parameter *n* where *n* is an input table.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

Example

Sample `_describe_extfn` API function code fragment that gets the scale of parameter 1:

```
if( cntxt->current_state > EXTFNAPIV4_STATE_ANNOTATION ) {
    a_sql_uint32 scale = 0;
    a_sql_int32 ret = 0;

    ret = ctx->describe_parameter_get( ctx, 1,
EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    &scale, sizeof(a_sql_uint32) );

    if( ret <= 0 ) {
        // Handle the error.
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL` attribute indicates whether or not the parameter is null. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_byte`

Description

True, if the value of a parameter can be NULL at the time of execution. For a TABLE parameter or parameter 0, the value is false.

Usage

Gets whether or not the specified parameter can be null during query execution.

Returns

On success, returns the `sizeof(a_sql_byte)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – Get error returned if the **describe_buffer** is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – Get error returned if the query processing phase is not greater than Plan Building.

Query Processing Phases

Valid in:

- Execution phase

Examples: EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL (Get)

Example procedure definitions, `_describe_extfn` API function code fragment, and SQL queries for getting **EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL** values.

Procedure Definition

Sample procedure definition used by the example queries in this topic:

```
CREATE PROCEDURE my_udf(IN p INT)
RESULT (x INT)
EXTERNAL NAME 'my_udf@myudflib';
```

API Function Code Fragment

Sample `_describe_extfn` API function code fragment used by the example queries in this topic:

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_byte can_be_null = 0;
        a_sql_int32 ret = 0;

        ret = cntxt->describe_parameter_get( cntxt, 1,
            EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
            &can_be_null,
            sizeof(a_sql_byte) );
    }
}
```

```
    if( ret <= 0 ) {  
        // Handle the error.  
    }  
}  
}
```

Example 1: Without NOT NULL

This example creates a table with a single integer column without the **NOT NULL** modifier specified. The correlated subquery passes in column *c1* from the table `has_nulls`. When the procedure `my_udf_describe` is called during the Execution state, the call to `describe_parameter_get` populates **can_be_null** with a value of 1.

```
CREATE TABLE has_nulls ( c1 INT );  
INSERT INTO has_nulls VALUES(1);  
INSERT INTO has_nulls VALUES(NULL);  
SELECT * from has_nulls WHERE (SELECT sum(my_udf.x) FROM  
my_udf(has_nulls.c1)) > 0;
```

Example 2: With NOT NULL

This example creates a table with a single integer column with the **NOT NULL** modifier specified. The correlated subquery passes in column *c1* from the table `no_nulls`. When the procedure `my_udf_describe` is called during the Execution state, the call to `describe_parameter_get` populates **can_be_null** with a value of 0.

```
CREATE TABLE no_nulls ( c1 INT NOT NULL);  
INSERT INTO no_nulls VALUES(1);  
INSERT INTO no_nulls VALUES(2);  
SELECT * from no_nulls WHERE (SELECT sum(my_udf.x) FROM  
my_udf(no_nulls.c1)) > 0;
```

Example 3: With a Constant

This example calls the procedure `my_udf` with a constant. When the procedure `my_udf_describe` is called, during the Execution state, the call to `describe_parameter_get` populates **can_be_null** with a value of 0.

```
SELECT * from my_udf(5);
```

Example 4: With a NULL

This example calls the procedure `my_udf` with a NULL. When the procedure `my_udf_describe` is called, during the Execution state, the call to `describe_parameter_get` populates **can_be_null** with a value of 1.

```
SELECT * from my_udf(NULL);
```

EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES Attribute (Get)

The **EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES** attribute returns the number of distinct values. Used in a `describe_parameter_get` scenario.

Data Type

```
a_v4_extfn_estimate
```


Description

Returns the estimated number of distinct values across all invocations. valid only for scalar parameters.

Usage

If this information is available, the UDF returns the estimated number of distinct values with 100% confidence. If the information is not available, the UDF returns an estimate of 0 with 0% confidence.

Returns

On success, returns the `sizeof(a_v4_extfn_estimate)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_v4_extfn_estimate`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than **Initial**.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the parameter is a **TABLE** parameter.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

Example

Sample `_describe_extfn` API function code fragment:

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_est.value = 0.0;
    desc_est.confidence = 0.0;

    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
        &desc_est, sizeof(a_v4_extfn_estimate) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES` attribute returns whether or not the parameter is constant. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_byte`

Description

True, if the parameter is a constant for the statement. Valid only for scalar parameters.

Usage

Returns 0 if the value of the specified parameter is not a constant; returns 1 if the value of the specified parameter is a constant.

Returns

On success, returns the `sizeof(a_sql_byte)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than `Initial`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the parameter is a `TABLE` parameter.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

Example

Sample `_describe_extfn` API function code fragment:

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
        &desc_byte, sizeof( a_sql_byte ) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE Attribute (Get)

The **EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE** attribute indicates the value of the parameter. Used in a `describe_parameter_get` scenario.

Data Type

`an_extfn_value`

Description

The value of the parameter if it is known at describe time. Valid only for scalar parameters.

Usage

Returns the value of the parameters.

Returns

On success, returns the `sizeof(an_extfn_value)` if the value is available, or:

- **EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE** – Value returned if the value is not constant.

On failure, returns one of the generic `describe_parameter` errors or:

- **EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH** – get error returned if the **describe_buffer** is not the size of `an_extfn_value`.
- **EXTFNAPIV4_DESCRIBE_INVALID_STATE** – get error returned if the phase is not greater than `Initial`.
- **EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER** – get error returned if the parameter is a `TABLE` parameter.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

Example

Sample `_describe_extfn` API function code fragment:

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    a_sql_int32 desc_rc;
    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,
        &arg,
        sizeof( an_extfn_value ) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS` attribute indicates the number of columns in the table. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_uint32`

Description

The number of columns in the table. Only valid for argument 0 and table arguments.

Usage

Returns the number of columns in the specified table argument. Argument 0 returns the number of columns in the result table.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `size of a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than `Initial`.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – get error returned if the parameter is not a `TABLE` parameter.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS` attribute indicates the number of rows in the table. Used in a `describe_parameter_get` scenario.

Data Type

`a_v4_extfn_estimate`

Description

The estimated number of rows in the table. Only valid for argument 0 and table arguments.

Usage

Returns the estimated number of rows in the specified table argument or result set with a confidence of 100%.

Returns

On success, returns the size of `a_v4_extfn_estimate`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_v4_extfn_estimate`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than **Initial**.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – get error returned if the parameter is not a **TABLE** parameter.

Query Processing Phases

Valid in:

- Annotation phase
- Query Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY` attribute indicates the order of rows in the table. Used in a `describe_parameter_get` scenario.

Data Type

`a_v4_extfn_orderby_list`

Description

The order of rows in the table. This property is only valid for argument 0 and table arguments.

Usage

This attribute allows the UDF code to:

- Determine if the input **TABLE** parameter has been ordered
- Declare that the result set is ordered

If the parameter number is 0, then the attribute refers to the outbound result set. If the parameter is > 0 and the parameter type is a table then the attribute refers to the input **TABLE** parameter.

The order is specified by the `a_v4_extfn_orderby_list`, which is a structure supporting a list of column ordinals and their associated ascending or descending property. If

the UDF sets the order by property for the outbound result set, the server is then able to perform order by optimizations. For example, if the UDF produced ascending order on the first result set column, the server will eliminate a redundant order by request on the same column.

If the UDF does not set the orderby property on the outbound result set, the server assumes the data is not ordered.

If the UDF sets the orderby property on the input **TABLE** parameter, the server guarantees data ordering for the input data. In this scenario, the UDF describes to the server that the input data must be ordered. If the server detects a runtime conflict it raises a SQL exception. For example, when the UDF describes that the first column of the input **TABLE** parameter must have ascending order and the SQL statement contains a descending clause, the server raises a SQL exception.

In the event that the SQL did not contain an ordering clause, the server automatically adds the ordering to ensure that input **TABLE** parameter is ordered as required.

Returns

If successful, returns the number of bytes copied from `a_v4_extfn_orderby_list`.

Query Processing States

Valid in:

- Annotation state
- Query optimization state

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY` attribute indicates that the UDF requires partitioning. Used in a `describe_parameter_get` scenario.

Data Type

`a_v4_extfn_column_list`

Description

UDF developers use `EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY` to programmatically declare that the UDF requires partitioning before invocation can proceed.

Usage

The UDF can inquire to the partition to enforce it, or to dynamically adapt the partitioning. It is the UDF's responsibility to allocate the `a_v4_extfn_column_list`, taking into consideration the total number of columns in the input table, and sending that data to the server.

Returns

On success, returns the size of `a_v4_extfn_column_list`. This value is equal to:

```
sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) *
number_of_partition_columns
```

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the buffer length is less than the expected size.

Query Processing Phases

Valid in:

- Query Optimization phase
- Plan Building phase
- Execution phase

Example

```
void UDF_CALLBACK my_tpf_proc_describe( a_v4_extfn_proc_context
*ctx )
{
    if( ctx->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_uint32 col_count = 0;
        a_sql_uint32 buffer_size = 0;
        a_v4_extfn_column_list *clist = NULL;

        col_count = 3;    // Set to the max number of possible pby
columns

        buffer_size = sizeof( a_v4_extfn_column_list ) + (col_count -
1) * sizeof( a_sql_uint32 );

        clist = (a_v4_extfn_column_list *)ctx->alloc( ctx,
buffer_size );

        clist->number_of_columns = 0;
        clist->column_indexes[0] = 0;
        clist->column_indexes[1] = 0;
        clist->column_indexes[2] = 0;

        args->r_api_rc = ctx->describe_parameter_get( ctx,
args->p3_arg_num,
EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
clist,
buffer_size );
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND` attribute indicates that the consumer requests rewind of an input table. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_byte`

Description

Indicates that the consumer wants to rewind an input table. Valid only for table input arguments. By default, this property is false.

Usage

The UDF queries this property to retrieve the true/false value.

Returns

On success, returns `sizeof(a_sql_byte)`.

On failure, returns one of the generic `describe_parameter` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the phase is not Optimization or Plan Building.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the UDF attempts to get this attribute on parameter 0.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – get error returned if the UDF attempts to get this attribute on a parameter that is not a table.

Query Processing Phases

Valid in:

- Optimization phase
- Plan Building phase

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND` attribute indicates if the parameter supports rewind. Used in a `describe_parameter_get` scenario.

Data Type

`a_sql_byte`

Description

Indicates whether a producer can support rewind. Valid only for table arguments.

You must also provide an implementation of the rewind table callback (`_rewind_extfn()`) if you plan on setting `DESCRIBE_PARM_TABLE_HAS_REWIND` to `true`. The server will fail to execute the UDF if the callback method is not provided.

Usage

The UDF asks if a table input argument supports rewind. As a prerequisite, the UDF must request rewind using `DESCRIBE_PARM_TABLE_REQUEST_REWIND` before you can use this property.

Returns

On success, returns `sizeof(a_sql_byte)`.

On failure, returns one of the generic `describe_parameter` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the **describe_buffer** is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than Annotation.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – get error returned if the UDF attempts to get this attribute on a parameter that is not a table.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the UDF attempts to get this attribute on the result table.

Query Processing Phases

Valid in:

- Optimization phase
- Plan Building phase
- Execution phase

EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` attribute lists unconsumed columns. Used in a `describe_parameter_get` scenario.

Data Type

`a_v4_extfn_column_list`

Description

The list of output table columns that are not going to be consumed by the server or the UDF.

For the output `TABLE` parameter, the UDF normally produces the data for all the columns, and the server consumes all the columns. The same holds true for the input `TABLE` parameter where the server normally produces the data for all the columns, and the UDF consumes all the columns.

However, in some cases the server, or the UDF, may not consume all the columns. The best practice in such a case is for the UDF to perform a **GET** for the output table on the describe

attribute `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS`. This action queries the server for the list of output table columns that are not going to be consumed by the server. The list can then be used by the UDF when populating the column data for the output table; that is, the UDF does not attempt to populate data for unused columns.

In summary, for the output table the UDF polls the list of unused columns. For the input table, the UDF pushes the list of unused columns.

Usage

The UDF asks the server if all the columns of the output table are going to be used. The UDF must allocate a `a_v4_extfn_column_list` that includes all the columns of the output table, and then must pass it to the server. The server then marks all the unprojected column ordinals as 1. The list returned by the server can be used while producing the data.

Returns

On success, returns the size of the column list: `sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) * number result columns`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the query processing phase is not greater than Plan Building.
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the `describe_buffer` is not large enough to hold the returned list.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the UDF attempts to get this attribute on an input table.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – get error returned if the UDF attempts to get this attribute on a parameter that is not a table.

Query Processing Phases

Valid in:

- Execution phase

***describe_parameter_set**

The `describe_parameter_set` v4 API method sets properties about a single parameter to the UDF.

Declaration

```
a_sql_int32 (SQL_CALLBACK *describe_parameter_set) (  
    a_v4_extfn_proc_context          *cntxt,  
    a_sql_uint32                     arg_num,  
    a_v4_extfn_describe_udf_type     describe_type,  
    const void                       *describe_buffer,  
    size_t                            describe_buffer_len );
```

Parameters

Parameter	Description
cntxt	The procedure context object.
arg_num	The ordinal of the TABLE parameter (0 is for the result table and 1 is for first input argument)
describe_type	A selector indicating what property to set.
describe_buffer	A structure that holds the describe information for the specified property to set on the server. The specific structure or data type is indicated by the describe_type parameter.
describe_buffer_length	The length in bytes of the describe_buffer .

Returns

On success, returns 0 or the number of bytes written to the **describe_buffer**. A value of 0 indicates that the server was unable to set the attribute, but no error condition occurred. If an error occurred, or no property was retrieved, this function returns one of the generic **describe_parameter** errors.

*Attributes for *describe_parameter_set*

Code showing the attributes for `describe_parameter_set`.

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

} a_v4_extfn_describe_parm_type;
```

EXTFNAPIV4_DESCRIBE_PARM_NAME Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_NAME` attribute indicates the parameter name. Used in a `describe_parameter_set` scenario.

Data Type

`char[]`

Description

The name of a parameter to a UDF.

Usage

If the UDF sets this property, the server compares the value with the name of the parameter supplied in the **CREATE PROCEDURE** statement. If the two values do not match, the server returns an error. This allows the UDF to ensure the **CREATE PROCEDURE** statement has the same parameter names as the UDF is expecting.

Returns

On success, returns the length of the parameter name.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not equal to `Annotation`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the parameter is the result table.
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF tries to reset the name.

Query Processing States

Valid in:

- `Annotation state`

EXTFNAPIV4_DESCRIBE_PARM_TYPE Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TYPE` attribute indicates the data type of the parameter. Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_data_type`

Description

The data type of a parameter to a UDF.

Usage

When the UDF sets this property, the server compares the value to the parameter type supplied in the **CREATE PROCEDURE** statement. If the two values do not match, the server returns an error. This check ensures that the **CREATE PROCEDURE** statement has the same parameter data types that the UDF expects.

Returns

On success, returns `sizeof(a_sql_data_type)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the `sizeof(a_sql_data_type)`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the query processing state is not equal to Annotation.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF tries to set the datatype of a parameter to something other than what it is already defined as.

Query Processing States

Valid in:

- Annotation state

EXTFNAPIV4_DESCRIBE_PARM_WIDTH Attribute (Set)

The **EXTFNAPIV4_DESCRIBE_PARM_WIDTH** attribute indicates the width of a parameter. Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_uint32`

Description

The width of a parameter to a UDF. `EXTFNAPIV4_DESCRIBE_PARM_WIDTH` applies only to scalar parameters. Parameter width is the amount of storage, in bytes, required to store a parameter of the associated data type.

- **Fixed length data types** – the bytes required to store the data.
- **Variable length data types** – the maximum length.
- **LOB data types** – the amount of storage required to store a handle to the data.
- **TIME data types** – the amount of storage required to store the encoded time.

Usage

This is a read-only property. The width is derived from the associated column data type. Once the data type is set, you cannot change the width.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the query processing state is not equal to `Annotation`.
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the specified parameter is a `TABLE` parameter. This includes parameter 0, or parameter *n*, where *n* is an input table.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF tries to reset the parameter width.

Query Processing States

Valid in:

- `Annotation` state

EXTFNAPIV4_DESCRIBE_PARM_SCALE Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_SCALE` attribute indicates the scale of a parameter.

Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_uint32`

Description

The scale of a parameter to a UDF. For arithmetic data types, parameter scale is the number of digits to the right of the decimal point in a number.

This attribute is invalid for:

- Nonarithmetic data types
- `TABLE` parameters

Usage

This is a read-only property. The scale is derived from the associated column data type. Once the data type is set, you cannot change the scale.

Returns

On success, returns `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not **Annotation**.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the specified parameter is a **TABLE** parameter. This includes parameter 0, or parameter *n*, where *n* is an input table.

Query Processing States

Valid in:

- Annotation state

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL Attribute (Set)

The **EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL** attribute returns whether or not the parameter is null. Using this attribute in a `describe_parameter_set` scenario returns an error.

Data Type

`a_sql_byte`

Description

True, if the value of a parameter can be **NULL** at the time of execution. For a **TABLE** parameter or parameter 0, the value is false.

Usage

This is a read-only property.

Returns

This is a read-only property, so all attempts to set result in an `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` error.

Query Processing States

Not applicable.

EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES Attribute (Set)

The **EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES** attribute returns the number of distinct values. Using this attribute in a `describe_parameter_set` scenario returns an error.

Data Type

`a_v4_extfn_estimate`

Description

Returns the estimated number of distinct values across all invocations. valid only for scalar parameters.

Usage

This is a read-only property.

Returns

This is a read-only property; all attempts to set result in an `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` error.

Query Processing States

Not applicable.

EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES` attribute returns whether or not the parameter is constant. Using this attribute in a `describe_parameter_set` scenario returns an error.

Data Type

`a_sql_byte`

Description

True, if the parameter is a constant for the statement. Valid only for scalar parameters.

Usage

This is a read-only property.

Returns

This is a read-only property; all attempts to set result in an `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` error.

Query Processing States

Not applicable.

EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE` attribute indicates the value of the parameter. Used in a `describe_parameter_set` scenario.

Data Type

`an_extfn_value`

Description

The value of the parameter if it is known at describe time. Valid only for scalar parameters.

Usage

This is a read-only property.

Returns

This is a read-only property; all attempts to set result in an `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` error.

Query Processing States

Not applicable.

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS` attribute indicates the number of columns in the table. Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_uint32`

Description

The number of columns in the table. Only valid for argument 0 and table arguments.

Usage

If the UDF sets this property, the server compares the value with the name of the parameter supplied in the **CREATE PROCEDURE** statement. If the two values do not match, the server returns an error. This allows the UDF to ensure the **CREATE PROCEDURE** statement has the same parameter names as the UDF is expecting.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the size of size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not `ANNOTATION`.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – set error returned if the parameter is not a `TABLE` parameter.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF tries to reset the number of columns of the specified table.

Query Processing States

Valid in:

- Annotation state

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS` attribute indicates the number of rows in the table. Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_a_v4_extfn_estimate`

Description

The estimated number of rows in the table. Only valid for argument 0 and table arguments.

Usage

The UDF sets this property for argument 0 if it estimates the number of rows in the result set. The server uses the estimate during optimization to make query processing decisions. You cannot set this value for an input table.

If you do not set a value, the server defaults to the number of rows specified by the **DEFAULT_TABLE_UDF_ROW_COUNT** option.

Returns

On success, returns `a_v4_extfn_estimate`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the size of `a_v4_extfn_estimate`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not **Optimization**.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – get error returned if the parameter is not a **TABLE** parameter.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – get error returned if the **TABLE** parameter is not the result table.
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – get error returned if the UDF tries to reset the number of columns of the specified table.

Query Processing States

Valid in:

- Query Optimization state

EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY` attribute indicates the order of rows in the table. Used in a `describe_parameter_set` scenario.

Data Type

`a_v4_extfn_orderby_list`

Description

The order of rows in the table. This property is only valid for argument 0 and table arguments.

Usage

This attribute allows the UDF code to:

- Determine if the input **TABLE** parameter has been ordered
- Declare that the result set is ordered.

If the parameter number is 0, then the attribute refers to the outbound result set. If the parameter is > 0 and the parameter type is a table then the attribute refers to the input **TABLE** parameter.

The order is specified by the `a_v4_extfn_orderby_list`, which is a structure supporting a list of column ordinals and their associated ascending or descending property. If the UDF sets the order by property for the outbound result set, the server is then able to perform order by optimizations. For example, if the UDF produced ascending order on the first result set column, the server will eliminate a redundant order by request on the same column.

If the UDF does not set the orderby property on the outbound result set, the server assumes the data is not ordered.

If the UDF sets the orderby property on the input **TABLE** parameter, the server guarantees data ordering for the input data. In this scenario, the UDF describes to the server that the input data must be ordered. If the server detects a runtime conflict it raises a SQL exception. For example, when the UDF describes that the first column of the input **TABLE** parameter must have ascending order and the SQL statement contains a descending clause, the server raises a SQL exception.

In the event that the SQL did not contain an ordering clause, the server automatically adds the ordering to ensure that input **TABLE** parameter is ordered as required.

Returns

If successful, returns the number of bytes copied from `a_v4_extfn_orderby_list`.

Query Processing States

Valid in:

- Annotation state
- Query optimization state

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY` attribute indicates that the UDF requires partitioning. Used in a `describe_parameter_set` scenario.

Data Type

`a_v4_extfn_column_list`

Description

UDF developers use **`EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY`** to programmatically declare that the UDF requires partitioning before invocation can proceed.

Usage

The UDF can inquire to the partition to enforce it, or to dynamically adapt the partitioning. The UDF must allocate the **`a_v4_extfn_column_list`**, taking into consideration the total number of columns in the input table, and sending that data to the server.

Returns

On success, returns the size of `a_v4_extfn_column_list`. This value is equal to:

```
sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) *  
number_of_partition_columns
```

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – Set error returned if the buffer length is less than the expected size.

Query Processing States

Valid in:

- Annotation state
- Query Optimization state

Example

```
void UDF_CALLBACK my_tpf_proc_describe( a_v4_extfn_proc_context  
*ctx )  
{  
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {  
        a_sql_int32 rc = 0;  
        a_v4_extfn_column_list pbcoll =  
        { 1, // 1 column in the partition by list  
          2 }; // column index 2 requires partitioning  
  
        // Describe partitioning for argument 1 (the table)  
        rc = ctx->describe_parameter_set(  
            ctx, 1,
```

```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
&pbcol,
sizeof(pbcol) );

if( rc == 0 ) {
ctx->set_error( ctx, 17000,
"Runtime error, unable set partitioning requirements for
column." );
}
}
}
}

```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND` attribute indicates that the consumer requests rewind of an input table. Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_byte`

Description

Indicates that the consumer wants to rewind an input table. Valid only for table input arguments. By default, this property is false.

Usage

If the UDF requires input table rewind capability, the UDF must set this property during Optimization.

Returns

On success, returns `sizeof(a_sql_byte)`.

On failure, returns one of the generic `describe_parameter` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not equal to Optimization.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the UDF attempts to set this attribute on parameter 0.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – set error returned if the UDF attempts to set this attribute on a parameter that is not a table.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF attempts to set this attribute to a value other than 0 or 1.

Query Processing States

Valid in:

- Optimization state

Example

In this example, when the function **my_udf_describe** is called during the Optimization state, the call to `describe_parameter_set` informs the producer of the table input parameter 1 that a rewind may be required.

Sample procedure definition:

```
CREATE PROCEDURE my_udf(IN t TABLE(c1 INT))
RESULT (x INT)
EXTERNAL NAME 'my_udf@myudflib';
```

Sample `_describe_extfn` API function code fragment:

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_byte rewind_required = 1;
        a_sql_int32 ret = 0;

        ret = cntxt->describe_parameter_set( cntxt, 1,
        EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
        &rewind_required,
        sizeof(a_sql_byte) );

        if( ret <= 0 ) {
            // Handle the error.
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND` attribute indicates if the parameter supports rewind. Used in a `describe_parameter_set` scenario.

Data Type

`a_sql_byte`

Description

Indicates whether a producer can support rewind. Valid only for table arguments.

You must also provide an implementation of the rewind table callback (`_rewind_extfn()`), if you plan on setting `DESCRIBE_PARM_TABLE_HAS_REWIND` to true. The server cannot execute the UDF if you do not provide the callback method.

Usage

A UDF sets this property during the Optimization state if it can provide rewind capability for its result table at no cost. If it is expensive for the UDF to provide rewind, do not set this property, or set it to 0. If set to 0, the server provides rewind support.

Returns

On success, returns `sizeof(a_sql_byte)`.

On failure, returns one of the generic `describe_parameter` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if the **describe_buffer** is not the size of `a_sql_byte`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not equal to `Optimization`.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – set error returned if the UDF attempts to set this attribute on a parameter that is not a table.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the specified argument is not the result table.
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – set error returned if the UDF attempts to set this attribute to a value other than 0 or 1.

Query Processing States

Valid in:

- Optimization state

EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS Attribute (Set)

The `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` attribute lists unconsumed columns. Used in a `describe_parameter_set` scenario.

Data Type

`a_v4_extfn_column_list`

Description

The list of output table columns that are not going to be consumed by the server or the UDF.

For the output `TABLE` parameter, the UDF normally produces the data for all the columns, and the server consumes all the columns. The same holds true for the input `TABLE` parameter where the server normally produces the data for all the columns, and the UDF consumes all the columns.

However, in some cases the server, or the UDF, may not consume all the columns. The best practice in such a case is that the UDF performs a **GET** for the output table on the `describe` attribute `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS`. This action queries the server for the list of output table columns which are not going to be consumed by the server. The list can then be used by the UDF when populating the column data for the output table; that is, the UDF skips populating data for unused columns.

In summary, for the output table the UDF polls the list of unused columns. For the input table, the UDF pushes the list of unused columns.

Usage

The UDF sets this property during Optimization if it is not going to use certain columns of the input TABLE parameter. The UDF must allocate a `a_v4_extfn_column_list` that includes all the columns of the output table, and then must pass it to the server. The server then marks all the un-projected column ordinals as 1. The server copies the list into its internal data structure.

Returns

On success, returns the size of the column list: `sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) * number result columns`.

On failure, returns one of the generic `describe_parameter` errors or:

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – set error returned if the state is not Optimization.
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if the UDF attempts to get this attribute on an input table.
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – set error returned if the UDF attempts to set this attribute on a parameter that is not a table.

Query Processing States

Valid in:

- Optimization state

***describe_udf_get**

The `describe_udf_get` v4 API method gets UDF properties from the server.

Declaration

```
a_sql_int32 (SQL_CALLBACK *describe_udf_get) (  
    a_v4_extfn_proc_context *cntxt,  
    a_v4_extfn_describe_udf_type describe_type,  
    void *describe_buffer,  
    size_t describe_buffer_len );
```

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object for this UDF.
<code>describe_type</code>	A selector indicating what property to retrieve.

Parameter	Description
describe_buffer	A structure that holds the describe information for the specified property to set on the server. The specific structure or data type is indicated by the describe_type parameter.
describe_buffer_length	The length in bytes of the describe_buffer .

Returns

On success, returns 0 or the number of bytes written to the **describe_buffer**. A value of 0 indicates that the server was unable to get the attribute but no error condition occurred. If an error occurred, or no property was retrieved, this function returns one of the generic `describe_udf` errors.

Attributes for `*describe_udf_get`

Code showing the attributes for `describe_udf_get`.

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extGetfn_describe_udf_type;
```

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS Attribute (Get)

The `EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS` attribute indicates the number of parameters. Used in a `describe_udf_get` scenario.

Data Type

`a_sql_uint32`

Description

The number of parameters supplied to the UDF.

Usage

Gets the number of parameters as defined in the **CREATE PROCEDURE** statement.

Returns

On success, returns the `sizeof(a_sql_uint32)`.

On failure, returns one of the generic `describe_udf` errors, or:

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – get error returned if the describe buffer is not the size of `a_sql_uint32`.
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – get error returned if the phase is not greater than Initial.

Query Processing Phases

- Annotation phase
- Query optimization phase
- Plan building phase
- Execution phase

***describe_udf_set**

The `describe_udf_set` v4 API method sets UDF properties on the server.

Declaration

```
a_sql_int32 (SQL_CALLBACK *describe_udf_set) (
    a_v4_extfn_proc_context *cntxt,
    a_v4_extfn_describe_udf_type describe_type,
    const void *describe_buffer,
    size_t describe_buffer_len );
```

Parameters

Parameter	Description
cntxt	The procedure context object for this UDF.
describe_type	A selector indicating what property to set.
describe_buffer	A structure that holds the describe information for the specified property to set on the server. The specific structure or data-type is indicated by the describe_type parameter.
describe_buffer_length	The length, in bytes, of describe_buffer .

Returns

On success, returns the number of bytes written to the **describe_buffer**. If an error occurs, or no property is retrieved, this function returns one of the generic `describe_udf` errors.

If an error occurs, or no property is retrieved, this function returns one of the generic `describe_udf` errors, or:

- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – set error returned if any of the **cntxt** or **describe_buffer** arguments are `NULL` or if **describe_buffer_length** is 0.
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – set error returned if there is a discrepancy between the requested attribute's size and the supplied **describe_buffer_length**.

*Attributes for *describe_udf_set*

Code showing the attributes for describe_udf_set.

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARAMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extGetfn_describe_udf_type;
```

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARAMS Attribute (Set)

The EXTFNAPIV4_DESCRIBE_UDF_NUM_PARAMS attribute indicates the number of parameters. Used in a describe_udf_set scenario.

Data Type

a_sql_uint32

Description

The number of parameters supplied to the UDF.

Usage

If the UDF sets this property, the server compares the value with the number of parameters supplied in the **CREATE PROCEDURE** statement. If the two values do not match, the server returns a SQL error. This allows the UDF to ensure the **CREATE PROCEDURE** statement has the same number of parameters expected by the UDF.

Returns

On success, returns the sizeof(a_sql_uint32).

On failure, returns one of the generic describe_udf errors, or:

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – Set error returned if the describe buffer is not the size of a_sql_uint32.
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – Set error returned if the state is not equal to Annotation.
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – set error returned if the UDF tries to reset the parameter datatype.

Query processing states

- Annotation state

Describe Column Type (a_v4_extfn_describe_col_type)

The `a_v4_extfn_describe_col_type` enumerated type selects the column property retrieved or set by the UDF.

Implementation

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    EXTFNAPIV4_DESCRIBE_COL_LAST
} a_v4_extfn_describe_col_type;
```

Members Summary

Member	Description
<i>EXTFNAPIV4_DESCRIBE_COL_NAME</i>	Column name (valid identifier).
<i>EXTFNAPIV4_DESCRIBE_COL_TYPE</i>	Column data type.
<i>EXTFNAPIV4_DESCRIBE_COL_WIDTH</i>	String width (precision for NUMERIC).
<i>EXTFNAPIV4_DESCRIBE_COL_SCALE</i>	Scale for NUMERIC.
<i>EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL</i>	True, if a column can be NULL.
<i>EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES</i>	Estimated number of distinct values in the column.
<i>EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE</i>	True, if column is unique within the table.
<i>EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT</i>	True, if column is constant for statement lifetime.
<i>EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE</i>	The value of a parameter, if known at describe time.

Member	Description
<i>EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER</i>	True, if column is needed by the consumer of the table.
<i>EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE</i>	The minimum value for the column (if known).
<i>EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE</i>	The maximum value for the column (if known).
<i>EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT</i>	The result column values are a subset of columns from an input table.
<i>EXTFNAPIV4_DESCRIBE_COL_LAST</i>	First illegal value for v4 API. Out-of-band value.

Describe Parameter Type (a_v4_extfn_describe_parm_type)

The `a_v4_extfn_describe_parm_type` enumerated type selects the parameter property retrieved or set by the UDF.

Implementation

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

    EXTFNAPIV4_DESCRIBE_PARM_LAST
} a_v4_extfn_describe_parm_type;
```

Members Summary

Member	Description
<i>EXTFNAPIV4_DESCRIBE_PARM_NAME</i>	Parameter name (valid identifier).
<i>EXTFNAPIV4_DESCRIBE_PARM_TYPE</i>	Data type.
<i>EXTFNAPIV4_DESCRIBE_PARM_WIDTH</i>	String width (precision for NUMERIC).
<i>EXTFNAPIV4_DESCRIBE_PARM_SCALE</i>	Scale for NUMERIC.
<i>EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL</i>	True, if the value can be NULL.
<i>EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES</i>	Estimated number of distinct values across all invocations.
<i>EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT</i>	True, if parameter is a constant for the statement.
<i>EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE</i>	The value of a parameter, if known at describe time.
These selectors can retrieve or set properties of a TABLE parameter. These enumerator values cannot be used with scalar parameters:	
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS</i>	The number of columns in the table.
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS</i>	Estimated number of rows in the table.
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY</i>	The order of rows in a table.
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY</i>	The partitioning; use <i>number_of_columns=0</i> for ANY.
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND</i>	True, if the consumer wants the ability rewind the input table.
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND</i>	Return true, if the producer supports rewind.

Member	Description
<i>EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS</i>	The list of output table columns that are not going to be consumed by the server or the UDF.
<i>EXTFNAPIV4_DESCRIBE_PARM_LAST</i>	First illegal value for v4 API. Out-of-band value.

Describe Return (a_v4_extfn_describe_return)

The `a_v4_extfn_describe_return` enumerated type provides a return value, when `a_v4_extfn_proc_context.describe_xxx_get()` or `a_v4_extfn_proc_context.describe_xxx_set()` does not succeed.

Implementation

```
typedef enum a_v4_extfn_describe_return {
    EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE           = 0,    // the specified operation has no
meaning either for this attribute or in
the current context.
    EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH   = -1,    // the provided buffer size
does not match the required length or the
length is insufficient.
    EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER      = -2,    // the provided parameter number
is invalid
    EXTFNAPIV4_DESCRIBE_INVALID_COLUMN        = -3,    // the column number is invalid
for this TABLE parameter
    EXTFNAPIV4_DESCRIBE_INVALID_STATE         = -4,    // the describe method call is not
valid in the present state
    EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE     = -5,    // the attribute is known but not
appropriate for this object
    EXTFNAPIV4_DESCRIBE_UNKNOWN_ATTRIBUTE     = -6,    // the identified attribute is
not known to this server version
    EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER   = -7,    // the specified parameter is
not a TABLE parameter (for describe_col_get()
or set())
    EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE = -8,    // the specified attribute
value is illegal
    EXTFNAPIV4_DESCRIBE_LAST                  = -9
} a_v4_extfn_describe_return;
```

Members Summary

Member	Re- turn Value	Description
<i>EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE</i>	0	The specified operation has no meaning either for this attribute or in the current context.

Member	Return Value	Description
<i>EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH</i>	-1	The provided buffer size does not match the required length, or the length is insufficient.
<i>EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER</i>	-2	The provided parameter number is invalid.
<i>EXTFNAPIV4_DESCRIBE_INVALID_COLUMN</i>	-3	The column number is invalid for this TABLE parameter.
<i>EXTFNAPIV4_DESCRIBE_INVALID_STATE</i>	-4	The describe method call is invalid in the present state.
<i>EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE</i>	-5	The attribute is known but not appropriate for this object.
<i>EXTFNAPIV4_DESCRIBE_UNKNOWN_ATTRIBUTE</i>	-6	The identified attribute is not known to this server version.
<i>EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER</i>	-7	The specified parameter is not a TABLE parameter (for <code>describe_col_get()</code> or <code>describe_col_set()</code>).
<i>EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE</i>	-8	The specified attribute value is illegal.
<i>EXTFNAPIV4_DESCRIBE_LAST</i>	-9	First illegal value for v4 API.

Description

The return value of `a_v4_extfn_proc_context.describe_xxx_get()` and `a_v4_extfn_proc_context.describe_xxx_set()` is a signed integer. If the result is positive, the operation succeeds, and the value is the number of bytes copied. If the

return value is less or equal to zero, the operation does not succeed, and the return value is one of the `a_v4_extfn_describe_return` values.

Describe UDF Type (`a_v4_extfn_describe_udf_type`)

Use the `a_v4_extfn_describe_udf_type` enumerated type to select the logical property the UDF retrieves or sets.

Implementation

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extfn_describe_udf_type;
```

Members Summary

Member	Description
<i>EXTFNAPIV4_DE- SCRIBE_UDF_NUM_PARMS</i>	The number of parameters supplied to the UDF.
<i>EXTFNAPIV4_DE- SCRIBE_UDF_LAST</i>	Out-of-band value.

Description

The `a_v4_extfn_proc_context.describe_udf_get()` method is used by the UDF to retrieve properties, and the `a_v4_extfn_proc_context.describe_udf_set()` method is used by the UDF to set properties about the UDF as a whole. The `a_v4_extfn_describe_udf_type` enumerator selects the logical property the UDF retrieves or sets.

Execution State (`a_v4_extfn_state`)

The `a_v4_extfn_state` enumerated type represents the query processing phase of a UDF.

Implementation

```
typedef enum a_v4_extfn_state {
    EXTFNAPIV4_STATE_INITIAL, // Server initial state,
    not used by UDF
    EXTFNAPIV4_STATE_ANNOTATION, // Annotating parse
    tree with UDF reference
    EXTFNAPIV4_STATE_OPTIMIZATION, // Optimizing
    EXTFNAPIV4_STATE_PLAN_BUILDING, // Building execution
    plan
    EXTFNAPIV4_STATE_EXECUTING, // Executing UDF and
    fetching results from UDF
    EXTFNAPIV4_STATE_LAST
} a_v4_extfn_state;
```

Members Summary

Member	Description
<i>EXTFNAPIV4_STATE_INITIAL</i>	Server initial phase. The only UDF method that is called during this query processing phase is <code>_start_extfn</code> .
<i>EXTFNAPIV4_STATE_ANNOTATION</i>	Annotating parse tree with UDF reference. The UDF is not invoked during this phase.
<i>EXTFNAPIV4_STATE_OPTIMIZATION</i>	Optimizing. The server calls the UDF's <code>_start_extfn</code> method, followed by the <code>_describe_extfn</code> function.
<i>EXTFNAPIV4_STATE_PLAN_BUILDING</i>	Building a query execution plan. The server calls the UDF's <code>_describe_extfn</code> function.
<i>EXTFNAPIV4_STATE_EXECUTING</i>	Executing UDF and fetching results from UDF. The server calls the <code>_describe_extfn</code> function before starting to fetch data from the UDF. The server then calls <code>_evaluate_extfn</code> to start the fetch cycle. During the fetch cycle, the server calls the functions defined in <code>a_v4_extfn_table_func</code> . When fetching finishes, the server calls the UDF's <code>_close_extfn</code> function.
<i>EXTFNAPIV4_STATE_LAST</i>	First illegal value for v4 API. Out-of-band value.

Description

The `a_v4_extfn_state` enumeration indicates which stage of UDF execution the server is in. When the server makes a transition from one phase to the next, the server informs the UDF it is leaving the previous phase by calling the UDF's `_leave_state_extfn` function. The server informs the UDF it is entering the new phase by calling the UDF's `_enter_state_extfn` function.

The query processing phase of a UDF restricts the operations that the UDF can perform. For example, in the Annotation phase, the UDF can retrieve the data types only for constant parameters.

External Function (a_v4_extfn_proc)

The server uses the `a_v4_extfn_proc` structure to call into the various entry points in the UDF. The server passes an instance of `a_v4_extfn_proc_context` to each of the functions.

Method Summary

Method	Description
<code>_start_extfn</code>	Allocates a structure and stores its address in the <code>_user_data</code> field in the <code>a_v4_extfn_proc_context</code> .
<code>_finish_extfn</code>	Deallocates a structure whose address was stored in the <code>user_data</code> field in the <code>a_v4_extfn_proc_context</code> .
<code>_evaluate_extfn</code>	Required function pointer to be called for each invocation of the function on a new set of argument values.
<code>_describe_extfn</code>	See <i>Describe API</i> on page 25.
<code>_enter_state_extfn</code>	The UDF can use this function to allocate structures.
<code>_leave_state_extfn</code>	The UDF can use this function to release memory or resources needed for the state.

_start_extfn

Use the `_start_extfn` v4 API method as an optional pointer to an initializer function, for which the only argument is a pointer to `a_v4_extfn_proc_context` structure.

Declaration

```
_start_extfn(  
a_v4_extfn_proc_context *  
)
```

Usage

Use the `_start_extfn` method to allocate a structure and store its address in the `_user_data` field in the `a_v4_extfn_proc_context`. This function pointer must be set to the null pointer if there is no need for any initialization.

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object.

finish_extfn

Use the `_finish_extfn` v4 API method as an optional pointer to a shutdown function, for which the only argument is a pointer to a `_v4_extfn_proc_context`.

Declaration

```
_finish_extfn(
    a_v4_extfn_proc_context *cntxt,
)
```

Usage

The `_finish_extfn` API deallocates a structure for which the address was stored in the `user_data` field in the `a_v4_extfn_proc_context`. This function pointer must be set to the null pointer if there is no need for any cleanup.

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object.

evaluate_extfn

Use the `_evaluate_extfn` v4 API method as a required function pointer that is called for each invocation of the function on a new set of argument values.

Declaration

```
_evaluate_extfn(
    a_v4_extfn_proc_context *cntxt,
    void *args_handle
)
```

Usage

The `_evaluate_extfn` function must describe to the server how to fetch results by filling in the `a_v4_extfn_table_func` portion of the `a_v4_extfn_table` structure and use the `set_value` method on the context with argument zero to send this information to the server. This function must also inform the server of its output schema by filling in the `a_v4_extfn_value_schema` of the `a_v4_extfn_table` structure before calling `set_value` on argument 0. It can access its input argument values via the `get_value` callback function. Both constant and nonconstant arguments are available to the UDF at this time.

Parameters

Parameter	Description
cntxt	The procedure context object.
args_handle	Handle to the arguments in the server.

_describe_extfn

`_describe_extfn` is called at the beginning of each state to allow the server to get and set logical properties. The UDF can do this by using the six describe methods (`describe_parameter_get`, `describe_parameter_set`, `describe_column_get`, `describe_column_set`, `describe_udf_get`, and `describe_udf_set`) in the `a_v4_proc_context` object.

See *Describe API* on page 25.

_enter_state_extfn

The UDF can implement the `_enter_state_extfn` v4 API method as an optional entry point to be notified whenever the UDF enters a new state.

Declaration

```
_enter_state_extfn(
    a_v4_extfn_proc_context *cntxt,
)
```

Usage

The UDF can use this notification to allocate structures.

Parameters

Parameter	Description
cntxt	The procedure context object.

_leave_state_extfn

The `_leave_state_extfn` v4 API method is an optional entry point the UDF can implement to receive a notification when the UDF moves out of a query processing state.

Declaration

```
_leave_state_extfn(
    a_v4_extfn_proc_context *cntxt,
)
```

Usage

The UDF can use this notification to release memory or resources needed for the state.

Parameters

Parameter	Description
cntxt	The procedure context object.

External Procedure Context (a_v4_extfn_proc_context)

Use the `a_v4_extfn_proc_context` structure to retain context information from the server and from the UDF.

Implementation

```
typedef struct a_v4_extfn_proc_context {
.
.
.
} a_v4_extfn_proc_context;
```

Method Summary

Re- turn Type	Method	Description
short	get_value	Gets input arguments to the UDF.
short	get_value_is_constant	Allows the UDF to ask whether a given argument is a constant.
short	set_value	Used by the UDF in either the <code>_evaluate_extfn</code> or <code>_describe_extfn</code> functions to describe to the server what its output will look like and to inform the server how to fetch results from the UDF.
a_sql_ uint32	get_is_cancelled	Call the get_is_cancelled callback every second or two to see if the user has interrupted the current statement.
short	set_error	Rolls back the current statement and generates an error.
void	log_message	Writes a message to the message log.
short	convert_value	Converts one data type to another.
short	get_option	Gets the value of a settable option.
void	alloc	Allocates a block of memory of length at least "len".
void	free	Free the memory allocated by alloc() for the specified lifetime.

Re- turn Type	Method	Description
a_sql_ uint32	describe_column_get	See <i>*describe_column_get</i> on page 26.
a_sql_ uint32	describe_column_set	See <i>*describe_column_set</i> on page 40.
a_sql_ uint32	describe_parameter_get	See <i>*describe_parameter_get</i> on page 55.
a_sql_ uint32	describe_parameter_set	See <i>*describe_parameter_set</i> on page 72.
a_sql_ uint32	describe_udf_get	See <i>*describe_udf_get</i> on page 86.
a_sql_ uint32	describe_udf_set	See <i>*describe_udf_set</i> on page 88.
short	open_result_set	Opens a result set for a table value.
short	close_result_set	Closes an open result set.
short	get_blob	Retrieves an input parameter that is a blob.
short	set_cannot_be_distrib- uted	Disables distribution at the UDF level even if the library is distributable.

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>_user_data</i>	void *	This data pointer can be filled in by any usage with whatever context data the external routine requires.
<i>_executionMode</i>	a_sql_uint32	Indicates the debug/trace level requested via the External_UDF_Execution_Mode option. This is a read-only field.
<i>current_state</i>	a_sql_uint32	The <i>current_state</i> attribute reflects the current execution mode of the context. This can be queried from functions such as <i>_describe_extfn</i> to determine what course of action to take.

Description

In addition to retaining context information from the server and the UDF, the structure `a_v4_extfn_proc_context` allows the UDF to call back into the server to perform

certain actions. The UDF can store private data in this structure in the `_user_data` member. An instance of this structure gets passed to the functions in the `a_v4_extfn_proc` method by the server. User data is not maintained until after the server reaches the Annotation state.

get_value

Use the `get_value` v4 API method to obtain the values of input arguments sent to the UDF in a SQL query.

Declaration

```
short get_value(  
    void *          arg_handle,  
    a_sql_uint32   arg_num,  
    an_extfn_value *value  
)
```

Usage

The `get_value` API is used in an evaluation method to retrieve the value of each input argument to the UDF. For narrow argument data types (>32K), a call to `get_value` is sufficient to retrieve the entire argument value.

The `get_value` API can be called from any API that has access to the `arg_handle` pointer. This includes API functions that take `a_v4_table_context` as a parameter. The `a_v4_table_context` has an `args_handle` member variable that can be used for this purpose.

For all fixed-length data types, the data is available in the returned value and no further calls are necessary to obtain all of the data. The producer can decide what the maximum length is that is returned entirely in the call to `get_value` method. All fixed length data types should be guaranteed to fit in a single contiguous buffer. For variable-length data, the limit is producer-dependant.

For nonfixed-length data types, and depending on the length of the data, a blob may need to be created using the `get_blob` method to get the data. You can use the macro **EXTFN_IS_INCOMPLETE** on the value returned by `get_value` to determine whether a blob object is required. If **EXTFN_IS_INCOMPLETE** evaluates to true, a blob is required.

For input arguments that are tables, the type is **AN_EXTFN_TABLE**. For this type of argument, you must create a result set using the `open_result_set` method to read values in from the table.

If a UDF requires the value of an argument prior to the `_evaluate_extfn` API being called, then the UDF should implement the `_describe_extfn` API. From the `_describe_extfn` API, the UDF can obtain the value of constant expressions using the `describe_parameter_get` method.

Parameters

Parameter	Description
arg_handle	A context pointer provided by the consumer.
arg_num	The index of the argument to get a value for. The argument index starts at 1.
value	The value of the specified argument.

Returns

1 if successful, 0 otherwise.

an_extfn_value Structure

The **an_extfn_value** structure represents the value of an input argument returned by the `get_value` API.

This code shows the declaration of the **an_extfn_value** structure:

```
short typedef struct an_extfn_value {
    void*      data;
    a_sql_uint32  piece_len,
    an_extfn_value *value {
        a_sql_uint32  total_len;
        a_sql_uint32  remain_len;
    } len;
    a_sql_data_type  type;
} an_extfn_value;
```

This table describes what the returned values of **an_extfn_value** object look like after calling the `get_value` method:

Value Returned by <code>get_value</code> API	EXTFN_IS_IN COMPLETE	total_len	piece_len	data
null	FALSE	0	0	null
empty string	FALSE	0	0	non-null
Size < MAX_UINT32	FALSE	actual	actual	non-null
size < MAX_UINT32	TRUE	actual	0	non-null
size >= MAX_UINT32	TRUE	MAX_UINT32	0	non-null

The type field of **an_extfn_value** contains the data type of the value. For UDFs that have tables as input arguments, the data type of that argument is **DT_EXTFN_TABLE**. For v4 Table UDFs, the `remain_len` field is not used.

get_value_is_constant

Use the `get_value_is_constant` v4 API method to determine whether the specified input argument value is a constant.

Declaration

```
short get_value_is_constant(  
    void *          arg_handle,  
    a_sql_uint32   arg_num,  
    an_extfn_value *value_is_constant  
)
```

Usage

The UDF can ask whether a given argument is a constant. This is useful for optimizing a UDF, for example, where work can be performed once during the first call to the `_evaluate_extfn` function, rather than for every evaluation call.

Parameters

Parameter	Description
arg_handle	Handle the arguments in the server.
arg_num	The index value of the input argument being retrieved. Index values are 1..N.
value_is_constant	Out parameter for storing is constant.

Returns

1 if successful, 0 otherwise.

set_value

Use the `set_value` v4 API method to describe to the consumer how many columns the result set has and how data should be read.

Declaration

```
short set_value(  
    void *          arg_handle,  
    a_sql_uint32   arg_num,  
    an_extfn_value *value  
)
```

Usage

This method is used by the UDF in the `_evaluate_extfn` API. The UDF must call the `set_value` method to tell the consumer how many columns are in the result set and what set of `a_v4_extfn_table_func` functions the UDF supports.

For the `set_value` API, the UDF provides an appropriate **arg_handle** pointer via the `_evaluate_extfn` API, or from the **args_handle** member of `a_v4_extfn_table_context` structure.

The **value** argument for the `set_value` method must be of type `DT_EXTFN_TABLE` for v4 Table UDFs.

Parameters

Parameter	Description
arg_handle	A context pointer provided by the consumer.
arg_num	The index of the argument to set a value for. The only supported argument is 0.
value	The value of the specified argument.

Returns

1 if successful, 0 otherwise.

get_is_cancelled

Use the `get_is_cancelled` v4 API method to determine whether the statement has been cancelled.

Declaration

```
short get_is_cancelled(
    a_v4_extfn_proc_context *    cntxt,
)
```

Usage

If a UDF entry point is performing work for an extended period of time (many seconds), it should, if possible, call the `get_is_cancelled` callback every second or two to see if the user has interrupted the current statement. If the statement has been interrupted, a nonzero value is returned and the UDF entry point should then immediately return. Call the `_finish_extfn` function to perform necessary cleanup. Do not subsequently call any other UDF entry points.

Parameters

Parameter	Description
cntxt	The procedure context object.

Returns

A nonzero value, if the statement is interrupted.

set_error

Use the `set_error` v4 API method to communicate an error back to the server and eventually to the user.

Declaration

```
void set_error(
    a_v4_extfn_proc_context *      cntxt,
    a_sql_uint32                  error_number,
    const char                     *error_desc_string
)
```

Usage

Call the `set_error` API, if a UDF entry point encounters an error that should send an error message to the user and shut down the current statement. When called, `set_error` API rolls back the current statement and the user sees "Error raised by user-defined function: <error_desc_string>". The `SQLCODE` is the negated form of the supplied <error_number>.

To avoid collisions with existing error codes, UDFs should generate error numbers between 17000 and 99999. If a number outside this range is provided, the statement is still rolled back, but the error message is "Invalid error raised by user-defined function: (<error_number>) <error_desc_string>" with a `SQLCODE` of -1577. The maximum length of **error_desc_string** is 140 characters.

After a call to `set_error` is made, the UDF entry point should immediately perform a return; eventually the `_finish_extfn` function is called to perform necessary cleanup. Do not subsequently call any other UDF entry points.

Parameters

Parameter	Description
cntxt	The procedure context object
error_number	The error number to set

Parameter	Description
error_desc_string	The message string to use

log_message

Use the `log_message` v4 API method to send a message to the server's message log.

Declaration

```
short log_message (
    const char      *msg,
    short           msg_length
)
```

Usage

The `log_message` method writes a message to the message log. The message string must be a printable text string no longer than 255 bytes; longer messages may be truncated.

Parameters

Parameter	Description
msg	The message string to log
msg_length	The length of the message string

convert_value

Use the `convert_value` v4 API method to convert data types.

Declaration

```
short convert_value (
    an_extfn_value *input,
    an_extfn_value *output
)
```

Usage

. The primary use of the `convert_value` API is the converting between `DT_DATE`, `DT_TIME`, and `DT_TIMESTAMP`, and `DT_TIMESTAMP_STRUCT`. An input and output `an_extfn_value` is passed to the function.

Input Parameters

Parameter	Description
an_extfn_value.data	Input data pointer

Parameter	Description
<code>an_extfn_value.total_len</code>	Length of input data
<code>an_extfn_value.type</code>	DT_ datatype of input

Output Parameters

Parameter	Description
<code>an_extfn_value.data</code>	UDF supplied output data point
<code>an_extfn_value.piece_len</code>	Maximum length of output data.
<code>an_extfn_value.total_len</code>	Server set length of converted
<code>an_extfn_value.type</code>	DT_ datatype of desired output

Returns

1 if successful, 0 otherwise.

get_option

The `get_option` v4 API method gets the value of a settable option.

Declaration

```
short get_option(
a_v4_extfn_proc_context * cntxt,
char *option_name,
an_extfn_value *output
)
```

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object
<code>option_name</code>	Name of the option to get
<code>output</code>	<ul style="list-style-type: none"> <code>an_extfn_value.data</code> – UDF supplied output data pointer <code>an_extfn_value.piece_len</code> – maximum length of output data <code>an_extfn_value.total_len</code> – server set length of converted output <code>an_extfn_value.type</code> – server set data type of value

Returns

1 if successful, 0 otherwise.

alloc

The `alloc` v4 API method allocates a block of memory.

Declaration

```
void*alloc(
    a_v4_extfn_proc_context *cntxt,
    size_t len
)
```

Usage

Allocates a block of memory of length at least `len`. The returned memory is 8-byte aligned.

Tip: Use the `alloc()` method as your only means of memory allocation, which allows the server to keep track of how much memory is used by external routines. The server can adapt other memory users, track leaks, and provide improved diagnostics and monitoring.

Memory tracking is enabled only when `external_UDF_execution_mode` is set to a value of 1 or 2 (validation mode or tracing mode).

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object
<code>len</code>	The length, in bytes, to allocate

free

The `free` v4 API method frees an allocated block of memory.

Declaration

```
void free(
    a_v4_extfn_proc_context *cntxt,
    void *mem
)
```

Usage

Frees the memory allocated by `alloc()` for the specified lifetime.

Memory tracking is enabled only when `external_UDF_execution_mode` is set to a value of 1 or 2 (validation mode or tracing mode).

Parameters

Parameter	Description
cntxt	The procedure context object
mem	Pointer to the memory allocated using the <code>alloc</code> method

open_result_set

The `open_result_set` v4 API method opens a result set for a table value.

Declaration

```
short open_result_set(
  a_v4_extfn_proc_context *cntxt,
  a_v4_extfn_table *table,
  a_v4_extfn_table_context **result_set
)
```

Usage

`open_result_set` opens a result set for a table value. A UDF can open a result set to read rows from an input parameter of type `DT_EXTFN_TABLE`. The server (or another UDF) can open a result set to read rows from the UDF.

Parameters

Parameter	Description
cntxt	The procedure context object
table	The table object on which to open a result set
result_set	An output parameter that is set to be an opened result set

Returns

1 if successful, 0 otherwise.

See the `fetch_block` and `fetch_into` v4 API method descriptions for examples of the use of `open_result_set`.

close_result_set

The `close_result_set` v4 API method closes an open result set.

Declaration

```
short close_result_set(
  a_v4_extfn_proc_context *cntxt,
```



```
a_v4_extfn_table_context *result_set
)
```

Usage

You can only use `close_result_set` once per result set.

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object
<code>result_set</code>	The result set to close

Returns

1 if successful, 0 otherwise.

get_blob

Use the `get_blob` v4 API method to retrieve an input blob parameter.

Declaration

```
short get_blob(
    void                *arg_handle,
    a_sql_uint32        arg_num,
    a_v4_extfn_blob    **blob
)
```

Usage

Use `get_blob` to retrieve a blob input parameter after calling `get_value()`. Use the macro `EXTFN_IS_INCOMPLETE` to determine if a blob object is required to read the data for the value returned from `get_value()`, if `piece_len < total_len`. The blob object is returned as an output parameter and is owned by the caller.

`get_blob` obtains a blob handle that can be used to read the contents of the blob. Call this method only on columns that contain blob objects.

Parameters

Parameter	Description
<code>arg_handle</code>	Handle to the arguments in the server
<code>arg_num</code>	The argument is a number 1..N
<code>blob</code>	Output argument containing the blob object

Returns

1 if successful, 0 otherwise.

set_cannot_be_distributed

The `set_cannot_be_distributed` v4 API method disables distributions at the UDF level, even if the distribution criteria are met at the library level.

Declaration

```
void set_cannot_be_distributed( a_v4_extfn_proc_context *cntxt)
```

Usage

In the default behavior, if the library is distributable, then the UDF is distributable. Use `set_cannot_be_distributed` in the UDF to push the decision to disable distribution to the server.

License Information (a_v4_extfn_license_info)

If you are a design partner, use the `a_v4_extfn_license_info` structure to define library-level license validations for your UDFs, including your company name, library version information, and an SAP-supplied license key.

Implementation

```
typedef struct an_extfn_license_info {
    short      version;
} an_extfn_license_info;

typedef struct a_v4_extfn_license_info {
    an_extfn_license_info version;

    const char      name[255];
    const char      info[255];
    void *          key;
} a_v4_extfn_license_info;
```

Data Member Summary

Data Member	Description
version	Internal use only. Must be set to 1.
name	Value the UDF sets as your company name.
info	Value the UDF sets for additional library information such as library version and build numbers.
key	(Design partners only) An SAP-supplied license key. The key is a 26-character array.

Optimizer Estimate (a_v4_extfn_estimate)

Use the `a_v4_extfn_estimate` structure to describe an estimate, which includes a value and a confidence level.

Implementation

```
typedef struct a_v4_extfn_estimate {
    double    value;
    double    confidence;
} a_v4_extfn_estimate;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>value</i>	double	The value for the estimate.
<i>confidence</i>	double	The confidence level associated with the estimate. The confidence varies from 0.0 to 1.0, with 0.0 meaning the estimate is invalid and 1.0 meaning the estimate is known to be true.

Order By List (a_v4_extfn_orderby_list)

Use the `a_v4_extfn_orderby_list` structure to describe the ORDER BY property of a table.

Implementation

```
typedef struct a_v4_extfn_orderby_list {
    a_sql_uint32    number_of_elements;
    a_v4_extfn_order_el order_elements[1];    // there are
number_of_elements entries
} a_v4_extfn_orderby_list;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>number_of_elements</i>	a_sql_uint32	The number of entries
<i>order_elements[1]</i>	a_v4_extfn_order_el	The order of the elements

Description

There are *number_of_elements* entries, each with a flag indicating whether the element is ascending or descending, and a column index indicating the appropriate column in the associated table.

Partition By Column Number (a_v4_extfn_partitionby_col_num)

The `a_v4_extfn_partitionby_col_num` enumerated type represents the column number to allow the UDF to express **PARTITION BY** support similar to that of SQL support.

Implementation

```
typedef enum a_v4_extfn_partitionby_col_num {
    EXTFNAPIV4_PARTITION_BY_COLUMN_NONE = -1,          // NO PARTITION
    BY
    EXTFNAPIV4_PARTITION_BY_COLUMN_ANY = 0,           // PARTITION BY
    ANY
                                                    // + INTEGER representing a specific
    column ordinal
} a_v4_extfn_partitionby_col_num;
```

Members Summary

Member of a_v4_extfn_partitionby_col_num Enumerated Type	Value	Description
<i>EXTFNAPIV4_PARTITION_BY_COLUMN_NONE</i>	-1	NO PARTITION BY
<i>EXTFNAPIV4_PARTITION_BY_COLUMN_ANY</i>	0	PARTITION BY ANY positive integer representing a specific column ordinal
<i>Column Ordinal Number</i>	N > 0	Ordinal for the table column number to partition on

Description

This structure allows the UDF to programmatically describe the partitioning and the column to partition on.

Use this enumeration when populating the `a_v4_extfn_column_list` `number_of_columns` field. When describing partition by support to the server, the UDF sets the `number_of_columns` to one of the enumerated values, or to a positive integer representing the number of column ordinals listed. For example, to describe to the server that no partitioning is supported, create the structure as:

```
a_v4_extfn_column_list nopby = {
    EXTFNAPIV4_PARTITION_BY_COLUMN_NONE,
```

```
0
};
```

The `EXTFNAPIV4_PARTITION_BY_COLUMN_ANY` member informs the server that the UDF supports any form of partitioning.

To describe a set of ordinals to partition on, create the structure as:

```
a_v4_extfn_column_list nopby = {
2,
3, 4
};
```

This describes a partition by over 2 columns whose ordinals are 3 and 4.

Note: This example is for illustrative purposes only and is not legal code. The caller must allocate the structure accordingly with room for 3 integers.

Row (a_v4_extfn_row)

Use the `a_v4_extfn_row` structure to represent the data in a single row.

Implementation

```
/* a_v4_extfn_row - */
typedef struct a_v4_extfn_row {
    a_sql_uint32 *row_status;
    a_v4_extfn_column_data *column_data;
} a_v4_extfn_row;
```

Data Members and Data Types Summary

Data Member	Data Type	Description
<code>row_status</code>	<code>a_sql_uint32 *</code>	The status of the row. Set to 1 for existing rows and 0 otherwise.
<code>column_data</code>	<code>a_v4_extfn_column_data *</code>	An array of column data for the row.

Description

The row structure contains information for a specific row of columns. This structure defines the status of an individual row and includes a pointer to the individual columns within the row. The row status is a flag that indicates the existence of a row. The row status flag can be altered by nested fetch calls without requiring manipulation of the row block structure.

The `row_status` flag set as 1 indicates that the row is available and can be included in the result set. The `row_status` set as 0 means the row should be ignored. This is useful when the TPF is acting as a filter because TPF may pass through rows of an input table to the result set, but it may also want to skip certain rows, which it can do by setting a status of 0 for those rows.

Row Block (a_v4_extfn_row_block)

Use the `a_v4_extfn_row_block` structure to represent the data in a block of rows.

Implementation

```

/* a_v4_extfn_row_block - */
typedef struct a_v4_extfn_row_block {
    a_sql_uint32      max_rows;
    a_sql_uint32      num_rows;
    a_v4_extfn_row    *row_data;
} a_v4_extfn_row_block;

```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>max_rows</i>	<code>a_sql_uint32</code>	The maximum number of rows this row block can handle
<i>num_rows</i>	<code>a_sql_uint32</code>	Must be less than or equal to the maximum of rows the row block contains
<i>row_data</i>	<code>a_v4_extfn_row *</code>	The row data vector

Description

The row block structure is utilized by the `fetch_into` and `fetch_block` methods to allow the production and consumption of data. The allocator sets the maximum number of rows. The producer incorrectly sets the number of rows. The data consumer should not attempt to read more than number of rows produced.

The owner of the `row_block` structure determines the value of `max_rows` data member. For example, when a table UDF is implementing `fetch_into`, the value of `max_rows` is determined by the server as the number of rows that can fit into 128K of memory. However, when a table UDF is implementing `fetch_block`, the table UDF itself determines the value of `max_rows`.

Restrictions and Limitations

The value for the both the `num_rows` and `max_rows` is > 0 . The `num_rows` must be \leq `max_rows`. The `row_data` field should not be NULL for a valid row block.

Table (a_v4_extfn_table)

Use the `a_v4_extfn_table` structure to represent how data is stored in a table and how the consumer fetches that data.

Implementation

```

typedef struct a_v4_extfn_table {
    a_v4_extfn_table_func *func;
}

```

```

    a_sql_uint32          number_of_columns;
} a_v4_extfn_table;

```

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>func</i>	a_v4_extfn_table_func *	This member holds a set of function pointers that the consumer uses to fetch result data
<i>number_of_columns</i>	a_sql_uint32 *	The number of columns in the table

Table Context (a_v4_extfn_table_context)

The a_v4_extfn_table_context structure represents an open result set over a table.

Implementation

```

typedef struct a_v4_extfn_table_context {
//    size_t struct_size;

    /* fetch_into() - fetch into a specified row_block. This entry point
       is used when the consumer has a transfer area with a specific format.
       The fetch_into() function will write the fetched rows into the provided row block.
    */
    short (UDF_CALLBACK *fetch_into)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block *);

    /* fetch_block() - fetch a block of rows. This entry point is used
       when the consumer does not need the data in a particular format. For example,
       if the consumer is reading a result set and formatting it as HTML, the consumer
       does not care how the transfer area is layed out. The fetch_block() entry point is
       more efficient if the consumer does not need a specific layout.

       The row_block parameter is in/out. The first call should point to a NULL row
       block.
       The fetch_block() call sets row_block to a block that can be consumed, and this
       block
       should be passed on the next fetch_block() call.
    */
    short (UDF_CALLBACK *fetch_block)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **row_block);

    /* rewind() - this is an optional entry point. If NULL, rewind is not supported.
       Otherwise,
       the rewind() entry point restarts the result set at the beginning of the table.
    */
    short (UDF_CALLBACK *rewind)(a_v4_extfn_table_context *);

    /* get_blob() - If the specified column has a blob object, return it. The blob
       is returned as an out parameter and is owned by the caller. This method should
       only be called on a column that contains a blob. The helper macro
       EXTFN_COL_IS_BLOB can
       be used to determine whether a column contains a blob.
    */
    short (UDF_CALLBACK *get_blob)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_column_data *col,
a_v4_extfn_blob **blob);

    /* The following fields are reserved for future use and must be initialized to NULL.
    */
    void *reserved1_must_be_null;
    void *reserved2_must_be_null;
    void *reserved3_must_be_null;

```

Using In-Database Analytics in Applications

```

void *reserved4_must_be_null;
void *reserved5_must_be_null;

a_v4_extfn_proc_context *proc_context;
void *args_handle; // use in
a_v4_extfn_proc_context::get_value() etc.
a_v4_extfn_table *table;
void *user_data;
void *server_internal_use;

/* The following fields are reserved for future use and must be initialized to NULL.
*/
void *reserved6_must_be_null;
void *reserved7_must_be_null;
void *reserved8_must_be_null;
void *reserved9_must_be_null;
void *reserved10_must_be_null;

} a_v4_extfn_table_context;

```

Method Summary

Data Type	Method	Description
short	fetch_into	Fetch into a specified <code>row_block</code>
short	fetch_block	Fetch a block of rows
short	rewind	Restarts the result set at the beginning of the table
short	get_blob	Return a blob object, if the specified column has a blob object

Data Members and Data Types Summary

Data Member	Data Type	Description
<i>proc_context</i>	<code>a_v4_extfn_proc_context *</code>	A pointer to the procedure context object. The UDF can use this to set errors, log messages, cancel, and so on.
<i>args_handle</i>	<code>void *</code>	A handle to the arguments provided by the server.
<i>table</i>	<code>a_v4_extfn_table *</code>	Points to the open result set table. This is populated after <code>a_v4_extfn_proc_context open_result_set</code> has been called.
<i>user_data</i>	<code>void *</code>	This data pointer can be filled in by any usage with whatever context data the external routine requires.
<i>server_internal_use</i>	<code>void *</code>	Internal use only.

Description

The `a_v4_extfn_table_context` structure acts as a middle layer between the producer and the consumer to help manage the data, when the consumer and producer require separate formats.

A UDF can read rows from an input `TABLE` parameter using `a_v4_extfn_table_context`. The server or another UDF can read rows from the result table of a UDF using `a_v4_extfn_table_context`.

The server implements the methods of `a_v4_extfn_table_context`, which gives the server an opportunity to resolve impedance mismatches.

fetch_into

The `fetch_into` v4 API method fetches data into a specified row block.

Declaration

```
short fetch_into(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block *)
```

Usage

The `fetch_into` method is useful when the producer does not know how data should be arranged in memory. This method is used as an entry point when the consumer has a transfer area with a specific format. The `fetch_into()` function writes the fetched rows into the provided row block. This method is part of the `a_v4_extfn_table_context` structure.

Use `fetch_into` when the consumer owns the memory for the data transfer area and requests that the producer use this area. You use the `fetch_into` method when the consumer cares about how the data transfer area is set up and it is up to the producer to perform the necessary data copying into this area.

Parameters

Parameter	Description
<code>cntxt</code>	The table context object obtained from the <code>open_result_set</code> API
<code>row_block</code>	The row block object to fetch into

Returns

1 if successful, 0 otherwise.

If the UDF returns 1, the consumer knows that there are more rows left and the `fetch_into` method should be called again. However, a UDF returning a value of 0 indicates that there are no more rows and a call to the `fetch_into` method is unnecessary.

Consider the following procedure definition, which is an example of a TPF function that consumes an input parameter table and produces it as a result table. Both are instances of SQL values that are obtained and returned through the `get_value` and `set_value` v4 API methods, respectively.

```
CREATE PROCEDURE FETCH_EX( IN a INT, INT b TABLE( c1 INT ) )
    RESULT SET ( rc INT )
```

This procedure definition contains two table objects:

- The input TABLE parameter named b
- The return result set table

The following example shows how output tables are fetched from by the caller, in this case, the server. The server might decide to use the `fetch_into` method. Input tables are fetched from by the called entity, in this case the TPF. The TPF decides which fetch API to use.

```
SELECT rc from FETCH_EX( 1, TABLE( SELECT c1 from TABLE ) )
```

The example shows that prior to fetching/consuming from an input table, a table context must be established via the `open_result_set` API on the `a_v4_extfn_proc` structure. The `open_result_set` requires a table object, which can be obtained through the `get_value` API.

```
an_extfn_value    arg;
ctx->get_value( args_handle, 3, &arg );

if( arg.type != DT_EXTFN_TABLE ) {
    // handle error
}

a_v4_extfn_table_context    *rs = NULL;
a_v4_extfn_table            *inTable = arg.data;
ctx->open_result_set( ctx, inTable, &rs );
```

After the table context is created, the `rs` structure executes the `fetch_into` API and fetches the rows.

```
a_v4_extfn_row_block    *rb = // get a row block to hold a series of
INT values.
rs->fetch_into( rs, &rb ) // fetch the rows.
```

Prior to producing rows to a result table, a table object must be created and returned to the caller via the `set_value` API on the `a_v4_extfn_proc_context` structure.

This example shows that a table UDF must create an instance of the `a_v4_extfn_table` structure. Each invocation of the table UDF should return a separate instance of the `a_v4_extfn_table` structure. The table contains the state fields to keep track of the current row and the number of rows to generate. State for a table can be stored as a field of the instance.

```
typedef struct rg_table : a_v4_extfn_table {
    a_sql_uint32    rows_to_generate;
```

```

    a_sql_uint32      current_row;
} my_table;

```

In the following example, each time a row is produced, **current_row** is incremented until the number of rows to be generated is reached, when `fetch_into` returns false to indicate end-of-file. The consumer executes the `fetch_into` API implemented by the table UDF. As part of the call to the `fetch_into` method, the consumer provides the table context, as well as the row block to fetch into.

```

    rs->fetch_into( rs, &rb )

short UDF_CALLBACK my_table_func_fetch_into(
    a_v4_extfn_table_context *tctx,
    a_v4_extfn_row_block *rb)
/*****
{
    my_table *myTable = tctx->table;

    if( rgTable->current_row < rgTable->rows_to_generate ) {
        // Produce the row...
        rgTable->current_row++;
        return 1;
    }

    return 0;
}

```

fetch_block

The `fetch_block` v4 API method fetches a block of rows.

Declaration

```

short fetch_block(
    a_v4_extfn_table_context *cntxt,
    a_v4_extfn_row_block **row_block)

```

Usage

The `fetch_block` method is used as an entry point when the consumer does not need the data in a particular format. `fetch_block` requests that the producer create a data transfer area and provide a pointer to that area. The consumer owns the memory and takes responsibility for copying data from this area.

The `fetch_block` is more efficient if the consumer does not require a specific layout. The `fetch_block` call sets a `fetch_block` to a block that can be consumed, and this block should be passed on the next `fetch_block` call. This method is part of the `a_v4_extfn_table_context` structure.

Parameters

Parameter	Description
cntxt	The table context object.
row_block	An in/out parameter. The first call should always point to a NULL row_block .

When `fetch_block` is called and **row_block** points to NULL, the UDF must allocate a `a_v4_extfn_row_block` structure.

Returns

1 if successful, 0 otherwise.

If the UDF returns 1, the consumer knows that there are more rows left and calls the `fetch_block` method again. However, a UDF returning a value of 0 indicates that there are no more rows and a call to the `fetch_block` method is unnecessary.

Consider the following procedure definition, which is an example of a TPF function that consumes an input parameter table and produces it as a result table. Both are instances of SQL values that are obtained and returned through the `get_value` and `set_value` v4 API methods, respectively.

```
CREATE PROCEDURE FETCH_EX( IN a INT, INT b TABLE( c1 INT ) )
  RESULT SET ( rc INT )
```

This procedure definition contains two table objects:

- The input TABLE parameter named `b`
- The return result set table

The following example shows how output tables are fetched from by the caller, in this case, the server. The server might decide to use the `fetch_block` method. Input tables are fetched from by the called entity, in this case the TPF, which decides which fetch API to use.

```
SELECT rc from FETCH_EX( 1, TABLE( SELECT c1 from TABLE ) )
```

The example shows that prior to fetching/consuming from an input table, a table context must be established via the `open_result_set` API on the `a_v4_extfn_proc` structure. The `open_result_set` requires a table object, which can be obtained through the `get_value` API.

```
an_extfn_value arg;
ctx->get_value( args_handle, 3, &arg );

if( arg.type != DT_EXTFN_TABLE ) {
  // handle error
}
```

```

a_v4_extfn_table_context  *rs = NULL;
a_v4_extfn_table          *inTable = arg.data;
ctx->open_result_set( ctx, inTable, &rs );

```

After the table context is created, the `rs` structure executes the `fetch_block` API and fetches the rows.

```

a_v4_extfn_row_block      *rb = // get a row block to hold a series of
INT values.
rs->fetch_block( rs, &rb ) // fetch the rows.

```

Prior to producing rows to a result table, a table object must be created and returned to the caller via the `set_value` API on the `a_v4_extfn_proc_context` structure.

This example shows that a table UDF must create an instance of the `a_v4_extfn_table` structure. Each invocation of the table UDF should return a separate instance of the `a_v4_extfn_table` structure. The table contains the state fields to keep track of the current row and the number of rows to generate. State for a table can be stored as a field of the instance.

```

typedef struct rg_table : a_v4_extfn_table {
    a_sql_uint32          rows_to_generate;
    a_sql_uint32          current_row;
} my_table;

```

rewind

Use the `rewind` v4 API method to restart a result set at the beginning of the table.

Declaration

```

short rewind(
    a_v4_extfn_table_context      *cntxt,
)

```

Usage

Call the `rewind` method on an open result set to rewind the table to the beginning. If the UDF intends to rewind an input table, it must inform the producer during the state

EXTFNAPIV4_STATE_OPTIMIZATION using the

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND parameter.

`rewind()` is an optional entry point. If `NULL`, `rewind` is not supported. Otherwise, the `rewind()` entry point restarts the result set at the beginning of the table.

Parameters

Parameter	Description
<code>cntxt</code>	The table context object

Returns

1 if successful, 0 otherwise.

get_blob

Use the `get_blob` v4 API method to return a blob object from a specified column.

Declaration

```
short get_blob(  
a_v4_extfn_table_context *cntxt,  
a_v4_extfn_column_data *col,  
a_v4_extfn_blob **blob  
)
```

Usage

The blob is returned as an output parameter and is owned by the caller. Call this method only on a column that contains a blob.

Use the helper macro `EXTFN_COL_IS_BLOB` to determine whether a column contains a blob. This is the declaration of `EXTFN_COL_IS_BLOB` in the header file

`extfnapiv4.h`:

```
#define EXTFN_COL_IS_BLOB(c, n) (c[n].blob_handle != NULL)
```

Parameters

Parameter	Description
cntxt	The table context object
col	The column data pointer for which to get the blob
blob	On success, contains the blob object associated with the column

Returns

1 if successful, 0 otherwise.

Table Functions (a_v4_extfn_table_func)

The consumer uses the `a_v4_extfn_table_func` structure to retrieve results from the producer.

Implementation

```
typedef struct a_v4_extfn_table_func {  
// size_t struct_size;  
  
/* Open a result set. The UDF can allocate any resources needed  
for the result set.
```

```

*/
short (UDF_CALLBACK *_open_extfn)(a_v4_extfn_table_context *);

/* Fetch rows into a provided row block. The UDF should implement
this method if it does
not have a preferred layout for its transfer area.
*/
short (UDF_CALLBACK *_fetch_into_extfn)(a_v4_extfn_table_context
*, a_v4_extfn_row_block
*row_block);

/* Fetch a block that is allocated and configured by the UDF. The
UDF should implement this
method if it has a preferred layout of the transfer area.
*/
short (UDF_CALLBACK *_fetch_block_extfn)
(a_v4_extfn_table_context *, a_v4_extfn_row_block
**row_block);

/* Restart a result set at the beginning of the table. This is an
optional entry point.
*/
short (UDF_CALLBACK *_rewind_extfn)(a_v4_extfn_table_context *);

/* Close a result set. The UDF can release any resources
allocated for the result set.
*/
short (UDF_CALLBACK *_close_extfn)(a_v4_extfn_table_context *);

/* The following fields are reserved for future use and must be
initialized to NULL. */
void *_reserved1_must_be_null;
void *_reserved2_must_be_null;
} a_v4_extfn_table_func;

```

Method Summary

Method	Data Type	Description
<code>_open_extfn</code>	void	Called by the server to initiate row fetching by opening a result set. The UDF can allocate any resources needed for the result set.
<code>_fetch_into_extfn</code>	short	Fetch rows into a provided row block. The UDF implements this method, if it does not have a preferred layout for its transfer area.
<code>_fetch_block_extfn</code>	short	Fetch a block that is allocated and configured by the UDF. The UDF implements this method, if it has a preferred layout of the transfer area.

Method	Data Type	Description
<code>_rewind_extfn</code>	void	Optional function called by the server to restart the fetching from the beginning of the table.
<code>_close_extfn</code>	void	Called by the server to terminate row fetching by closing the result set. The UDF can release any resources allocated for the result set.
<code>_reserved1_must_be_null</code>	void	Reserved for future use. Must be initialized to NULL.
<code>_reserved1_must_be_null</code>	void	Reserved for future use. Must be initialized to NULL.

Description

The `a_v4_extfn_table_func` structure defines the methods used to fetch results from a table.

open_extfn

The server calls the `_open_extfn` v4 API method to initiate fetching of rows.

Description

```
void _open_extfn(
    a_v4_extfn_table_context *cntxt,
)
```

Usage

The UDF uses this method to open a result set and allocate any resources (for example, streams) needed for sending results to the server.

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object

fetch_into_extfn

The `_fetch_into_extfn` v4 API method fetches rows into a provided row block.

Description

```
short _fetch_into_extfn(
    a_v4_extfn_table_context *cntxt,
```



```
a_v4_extfn_row_block *row_block
)
```

Usage

The UDF should implement this method, if it does not have a preferred layout for its transfer area.

Parameters

Parameter	Description
cntxt	The procedure context object
row_block	The row block object to fetch into.

Returns

1 if successful, 0 otherwise.

fetch_block_extfn

The `_fetch_block_extfn` v4 API method fetches a block that is allocated and configured by the UDF.

Declaration

```
short _fetch_block_extfn(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **
)
```

Usage

The UDF should implement this method, if it has a preferred layout for its transfer area.

Parameters

Parameter	Description
cntxt	The procedure context object
row_block	The row block object to fetch into

Returns

1 if successful, 0 otherwise.

_rewind_extfn

The `_rewind_extfn` v4 API method restarts a result set at the beginning of the table.

Declaration

```
void _rewind_extfn(  
a_v4_extfn_table_context *cntxt,  
)
```

Usage

This function is an optional entry point. The UDF implements the `_rewind_extfn` method when the result table is rewound to the beginning. The UDF should consider implementing this method only if it can provide the rewind functionality in an efficient and cost-effective manner.

If a UDF chooses to implement the `_rewind_extfn` method, it should tell the consumer during the state **EXTFNAPIV4_STATE_OPTIMIZATION** by setting the **EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND** parameter for argument 0.

The UDF may decide not to provide the rewind functionality, in which case the server compensates and provides the functionality.

Note: The server can choose not to call the `_rewind_extfn` method to perform the rewind.

Parameters

Parameter	Description
<code>cntxt</code>	The procedure context object

Returns

No return value.

_close_extfn

The server calls the `_close_extfn` v4 API method to terminate fetching of rows.

Declaration

```
void _close_extfn(  
a_v4_extfn_table_context *cntxt,  
)
```

Usage

The UDF uses this method when fetching is complete to close a result set and release any resources allocated for the result set.

Parameters

Parameter	Description
cntxt	The procedure context object

Using SQL in Applications

This section provides information about using SQL in applications.

SQL statement execution in applications

The way you include SQL statements in your application depends on the application development tool and programming interface you use.

- **ADO.NET** – You can execute SQL statements using various ADO.NET objects. The `SACommand` object is one example:

```
SACommand cmd = new SACommand(
    "DELETE FROM Employees WHERE EmployeeID = 105", conn );
cmd.ExecuteNonQuery();
```

- **ODBC** – If you are writing directly to the ODBC programming interface, your SQL statements appear in function calls. For example, the following C function call executes a DELETE statement:

```
SQLExecDirect( stmt,
    "DELETE FROM Employees
    WHERE EmployeeID = 105",
    SQL_NTS );
```

- **JDBC** – If you are using the JDBC programming interface, you can execute SQL statements by invoking methods of the statement object. For example:

```
stmt.executeUpdate(
    "DELETE FROM Employees
    WHERE EmployeeID = 105" );
```

- **Embedded SQL** – If you are using embedded SQL, you prefix your C language SQL statements with the keyword `EXEC SQL`. The code is then run through a preprocessor before compiling. For example:

```
EXEC SQL EXECUTE IMMEDIATE
    'DELETE FROM Employees
    WHERE EmployeeID = 105';
```

- **Sybase Open Client** – If you use the Sybase Open Client interface, your SQL statements appear in function calls. For example, the following pair of calls executes a DELETE statement:

```
ret = ct_command( cmd, CS_LANG_CMD,
    "DELETE FROM Employees
    WHERE EmployeeID=105"
    CS_NULLTERM,
    CS_UNUSED);
ret = ct_send(cmd);
```

For more details about including SQL in your application, see your development tool documentation. If you are using ODBC or JDBC, consult the software development kit for those interfaces.

Applications inside the database server

In many ways, stored procedures and triggers act as applications or parts of applications running inside the database server. You can also use many of the techniques here in stored procedures.

Java classes in the database can use the JDBC interface in the same way as Java applications outside the server. This section discusses some aspects of JDBC.

Prepared statements

Each time a statement is sent to a database, the database server must perform the following steps:

- It must parse the statement and transform it into an internal form. This process is sometimes called *preparing* the statement.
- It must verify the correctness of all references to database objects by checking, for example, that columns named in a query actually exist.
- If the statement involves joins or subqueries, then the query optimizer generates an access plan.
- It executes the statement after all these steps have been carried out.

Reusing prepared statements can improve performance

If you use the same statement repeatedly, for example inserting many rows into a table, repeatedly preparing the statement causes a significant and unnecessary overhead. To remove this overhead, some database programming interfaces provide ways of using prepared statements. A *prepared* statement is a statement containing a series of placeholders. When you want to execute the statement, assign values to the placeholders, rather than prepare the entire statement over again.

Using prepared statements is useful when carrying out many similar actions, such as inserting many rows.

Generally, using prepared statements requires the following steps:

- **Prepare the statement** – In this step, you generally provide the statement with some placeholder character instead of the values.
- **Repeatedly execute the prepared statement** – In this step, you supply values to be used each time the statement is executed. The statement does not have to be prepared each time.
- **Drop the statement** – In this step, you free the resources associated with the prepared statement. Some programming interfaces handle this step automatically.

Do not prepare statements that are used only once

In general, you should not prepare statements if they are only executed once. There is a slight performance penalty for separate preparation and execution, and it introduces unnecessary complexity into your application.

In some interfaces, however, you do need to prepare a statement to associate it with a cursor.

The calls for preparing and executing statements are not a part of SQL, and they differ from interface to interface. Each of the SAP Sybase IQ programming interfaces provides a method for using prepared statements.

Prepared Statements Overview

This section provides a brief overview of how to use prepared statements. The general procedure is the same, but the details vary from interface to interface. Comparing how to use prepared statements in different interfaces illustrates this point.

You typically perform the following tasks to use a prepared statement:

1. Prepare the statement.
2. Bind the parameters that will hold values in the statement.
3. Assign values to the bound parameters in the statement.
4. Execute the statement.
5. Repeat steps 3 and 4 as needed.
6. Drop the statement when finished. In JDBC the Java garbage collection mechanism drops the statement.

Use a Prepared Statement in ADO.NET

You typically perform the following tasks to use a prepared statement in ADO.NET:

1. Create an `SACommand` object holding the statement:

```
SACommand cmd = new SACommand(
    "SELECT * FROM Employees WHERE Surname = ?", conn );
```

2. Declare data types for any parameters in the statement.

Use the `SACommand.CreateParameter` method.

```
SAParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Char;
param.Direction = ParameterDirection.Input;
param.Value = "Smith";
cmd.Parameters.Add(param);
```

3. Prepare the statement using the `Prepare` method.
4. Execute the statement:

```
SADataReader reader = cmd.ExecuteReader();
```

For an example of preparing statements using ADO.NET, see the source code in `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleWin32`.

Use a prepared statement in ODBC

You typically perform the following tasks to use a prepared statement in ODBC:

1. Prepare the statement using `SQLPrepare`.
2. Bind the statement parameters using `SQLBindParameter`.
3. Execute the statement using `SQLExecute`.
4. Drop the statement using `SQLFreeStmt`.

For an example of preparing statements using ODBC, see the source code in `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ODBCPrepare`.

Use a Prepared Statement in JDBC

You typically perform the following tasks to use a prepared statement in JDBC:

1. Prepare the statement using the `prepareStatement` method of the connection object. This returns a prepared statement object.
2. Set the statement parameters using the appropriate `setType` methods of the prepared statement object. Here, *Type* is the data type assigned.
3. Execute the statement using the appropriate method of the prepared statement object. For inserts, updates, and deletes this is the `executeUpdate` method.

For an example of preparing statements using JDBC, see the source code file `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCExample.java`.

Use a Prepared Statement in Embedded SQL

You typically perform the following tasks to use a prepared statement in embedded SQL:

1. Prepare the statement using the `EXEC SQL PREPARE` statement.
2. Assign values to the parameters in the statement.
3. Execute the statement using the `EXEC SQL EXECUTE` statement.
4. Free the resources associated with the statement using the `EXEC SQL DROP` statement.

Use a Prepared Statement in Open Client

You typically perform the following tasks to use a prepared statement in Open Client:

1. Prepare the statement using the `ct_dynamic` function, with a `CS_PREPARE` type parameter.
2. Set statement parameters using `ct_param`.
3. Execute the statement using `ct_dynamic` with a `CS_EXECUTE` type parameter.
4. Free the resources associated with the statement using `ct_dynamic` with a `CS_DEALLOC` type parameter.

Cursor usage

When you execute a query in an application, the result set consists of several rows. In general, you do not know how many rows the application is going to receive before you execute the query. Cursors provide a way of handling query result sets in applications.

The way you use cursors and the kinds of cursors available to you depend on the programming interface you use.

SAP Sybase IQ provides several system procedures to help determine what cursors are in use for a connection, and what they contain:

```
sa_list_cursors system procedure
sa_describe_cursor system procedure
sa_copy_cursor_to_temp_table system procedure
```

With cursors, you can perform the following tasks within any programming interface:

- Loop over the results of a query.
- Perform inserts, updates, and deletes on the underlying data at any point within a result set.

In addition, some programming interfaces allow you to use special features to tune the way result sets return to your application, providing substantial performance benefits for your application.

Cursors

A cursor is a name associated with a result set. The result set is obtained from a SELECT statement or stored procedure call.

A cursor is a handle on the result set. At any time, the cursor has a well-defined position within the result set. With a cursor you can examine and possibly manipulate the data one row at a time. SAP Sybase IQ cursors support forward and backward movement through the query results.

Cursor positions

Cursors can be positioned in the following places:

- Before the first row of the result set.
- On a row in the result set.
- After the last row of the result set.

Using SQL in Applications

Absolute row from start		Absolute row from end
0	Before first row	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		-3
$n - 1$		-2
n		-1
$n + 1$	After last row	0

The cursor position and result set are maintained in the database server. Rows are *fetched* by the client for display and processing either one at a time or a few at a time. The entire result set does not need to be delivered to the client.

Benefits of using cursors

Although server-side cursors are not required in database applications, they do provide several benefits. A server-side cursor is preferable to a client-side cursor for the following reasons:

- **Response time** – Server-side cursors do not require that the whole result set be assembled before the first row is fetched by the client. A client-side cursor requires that the entire result set be obtained and transferred to the client before the first row is fetched by the client.
- **Client-side memory** – For large result sets, obtaining the entire result set on the client side can lead to demanding memory requirements.
- **Concurrency control** – If you make updates to your data and do not use server-side cursors in your application, you must send separate SQL statements like UPDATE, INSERT, or DELETE to the database server to apply changes. This raises the possibility of concurrency problems if any corresponding rows in the database have changed since the result set was delivered to the client. As a consequence, updates by other clients may be lost.

Server-side cursors can act as pointers to the underlying data, permitting you to impose proper concurrency constraints on any changes made by the client by setting an appropriate isolation level.

Cursor principles

To use a cursor in ADO.NET, ODBC, JDBC, or Open Client, follow these general steps:

1. Prepare and execute a statement.

Execute a statement using the usual method for the interface. You can prepare and then execute the statement, or you can execute the statement directly.

With ADO.NET, only the `SACommand.ExecuteReader` method returns a cursor. It provides a read-only, forward-only cursor.

2. Test to see if the statement returns a result set.

A cursor is implicitly opened when a statement that creates a result set is executed. When the cursor is opened, it is positioned before the first row of the result set.

3. Fetch results.

Although simple fetch operations move the cursor to the next row in the result set, SAP Sybase IQ permits more complicated movement around the result set.

4. Close the cursor.

When you have finished with the cursor, close it to free associated resources.

5. Free the statement.

If you used a prepared statement, free it to reclaim memory.

The approach for using a cursor in embedded SQL differs from the approach used in other interfaces. Follow these general steps to use a cursor in embedded SQL:

1. Prepare a statement.

Cursors generally use a statement handle rather than a string. You need to prepare a statement to have a handle available.

2. Declare the cursor.

Each cursor refers to a single `SELECT` or `CALL` statement. When you declare a cursor, you state the name of the cursor and the statement it refers to.

3. Open the cursor.

For a `CALL` statement, opening the cursor executes the procedure up to the point where the first row is about to be obtained.

4. Fetch results.

Although simple fetch operations move the cursor to the next row in the result set, SAP Sybase IQ permits more complicated movement around the result set. How you declare the cursor determines which fetch operations are available to you.

5. Close the cursor.

When you have finished with the cursor, close it. This frees any resources associated with the cursor.

6. Drop the statement.

To free the memory associated with the statement, you must drop the statement.

Cursor positioning

When a cursor is opened, it is positioned before the first row. You can move the cursor position to an absolute position from the start or the end of the query results, or to a position relative to the current cursor position. The specifics of how you change cursor position, and what operations are possible, are governed by the programming interface.

The number of row positions you can fetch in a cursor is governed by the size of an integer. You can fetch rows numbered up to number 2147483646, which is one less than the value that can be held in an integer. When using negative numbers (rows from the end) you can fetch down to one more than the largest negative value that can be held in an integer.

You can use special positioned update and delete operations to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, an error is returned indicating that there is no corresponding cursor row.

Note: Inserts and some updates to asensitive cursors can cause problems with cursor positioning. SAP Sybase IQ does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear at all until the cursor is closed and opened again. With SAP Sybase IQ, this occurs if a work table had to be created to open the cursor.

The UPDATE statement may cause a row to move in the cursor. This happens if the cursor has an ORDER BY clause that uses an existing index (a work table is not created). Using STATIC SCROLL cursors alleviates these problems but requires more memory and processing.

Cursor behavior when opening cursors

You can configure the following aspects of cursor behavior when you open the cursor:

- **Isolation level** – You can explicitly set the isolation level of operations on a cursor to be different from the current isolation level of the transaction. To do this, set the `isolation_level` option.
- **Holding** – By default, cursors in embedded SQL close at the end of a transaction. Opening a cursor WITH HOLD allows you to keep it open until the end of a connection, or until you explicitly close it. ADO.NET, ODBC, JDBC, and Open Client leave cursors open at the end of transactions by default.

Row fetching through a cursor

The simplest way of processing the result set of a query using a cursor is to loop through all the rows of the result set until there are no more rows. You can accomplish this task by performing these steps:

1. Declare and open the cursor (embedded SQL), or execute a statement that returns a result set (ODBC, JDBC, Open Client) or SqlDataReader object (ADO.NET).
2. Continue to fetch the next row until you get a Row Not Found error.

3. Close the cursor.

The technique used to fetch the next row is dependent on the interface you use. For example:

- **ADO.NET** – Use the `SADataReader.Read` method.
- **ODBC** – `SQLFetch`, `SQLExtendedFetch`, or `SQLFetchScroll` advances the cursor to the next row and returns the data.
- **JDBC** – The next method of the `ResultSet` object advances the cursor and returns the data.
- **Embedded SQL** – The `FETCH` statement carries out the same operation.
- **Open Client** – The `ct_fetch` function advances the cursor to the next row and returns the data.

Multiple-row fetching

Multiple-row fetching should not be confused with prefetching rows. Multiple row fetching is performed by the application, while prefetching is transparent to the application, and provides a similar performance gain. Fetching multiple rows at a time can improve performance.

Multiple-row fetches

Some interfaces provide methods for fetching more than one row at a time into the next several fields in an array. Generally, the fewer separate fetch operations you execute, the fewer individual requests the server must respond to, and the better the performance. A modified `FETCH` statement that retrieves multiple rows is also sometimes called a *wide fetch*. Cursors that use multiple-row fetches are sometimes called *block cursors* or *fat cursors*.

Using multiple-row fetching

- In ODBC, you can set the number of rows that will be returned on each call to `SQLFetchScroll` or `SQLExtendedFetch` by setting the `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ROWSET_SIZE` attribute.
- In embedded SQL, the `FETCH` statement uses an `ARRAY` clause to control the number of rows fetched at a time.
- Open Client and JDBC do not support multi-row fetches. They do use prefetching.

Scrollable cursors

ODBC and embedded SQL provide methods for using scrollable cursors and dynamic scrollable cursors. These methods allow you to move several rows forward at a time, or to move backward through the result set.

The JDBC and Open Client interfaces do not support scrollable cursors.

Prefetching does not apply to scrollable operations. For example, fetching a row in the reverse direction does not prefetch several previous rows.

Cursors used to modify rows

Cursors can do more than just read result sets from a query. You can also modify data in the database while processing a cursor. These operations are commonly called *positioned* insert, update, and delete operations, or PUT operations if the action is an insert.

Not all query result sets allow positioned updates and deletes. If you perform a query on a non-updatable view, then no changes occur to the underlying tables. Also, if the query involves a join, then you must specify which table you want to delete from, or which columns you want to update, when you perform the operations.

Inserts through a cursor can only be executed if any non-inserted columns in the table allow NULL or have defaults.

If multiple rows are inserted into a value-sensitive (keyset driven) cursor, they appear at the end of the cursor result set. The rows appear at the end, even if they do not match the WHERE clause of the query or if an ORDER BY clause would normally have placed them at another location in the result set. This behavior is independent of programming interface. For example, it applies when using the embedded SQL PUT statement or the ODBC SQLBulkOperations function. The value of an AUTOINCREMENT column for the most recent row inserted can be found by selecting the last row in the cursor. For example, in embedded SQL the value could be obtained using `FETCH ABSOLUTE -1 cursor-name`. As a result of this behavior, the first multiple-row insert for a value-sensitive cursor may be expensive.

ODBC, JDBC, embedded SQL, and Open Client permit data manipulation using cursors, but ADO.NET does not. With Open Client, you can delete and update rows, but you can only insert rows on a single-table query.

Which table are rows deleted from?

If you attempt a positioned delete through a cursor, the table from which rows are deleted is determined as follows:

1. If no FROM clause is included in the DELETE statement, the cursor must be on a single table only.
2. If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.
3. If a FROM clause is included, and no table owner is specified, the table-spec value is first matched against any correlation names.
4. If a correlation name exists, the table-spec value is identified with the correlation name.
5. If a correlation name does not exist, the table-spec value must be unambiguously identifiable as a table name in the cursor.
6. If a FROM clause is included, and a table owner is specified, the table-spec value must be unambiguously identifiable as a table name in the cursor.

7. The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

Updatable statements

This section describes how clauses in the SELECT statement affect updatable statements and cursors.

Updatability of read-only statements

Specifying FOR READ ONLY in the cursor declaration, or including a FOR READ ONLY clause in the statement, renders the statement read-only. In other words, a FOR READ ONLY clause, or the appropriate read-only cursor declaration when using a client API, overrides any other updatability specification.

If the outermost block of a SELECT statement contains an ORDER BY clause, and the statement does not specify FOR UPDATE, then the cursor is read-only. If the SQL SELECT statement specifies FOR XML, then the cursor is read-only. Otherwise, the cursor is updatable.

Updatable statements and concurrency control

For updatable statements, SAP Sybase IQ provides both optimistic and pessimistic concurrency control mechanisms on cursors to ensure that a result set remains consistent during scrolling operations. These mechanisms are alternatives to using INSENSITIVE cursors or snapshot isolation, although they have different semantics and tradeoffs.

The specification of FOR UPDATE can affect whether a cursor is updatable. However, in SAP Sybase IQ, the FOR UPDATE syntax has no other effect on concurrency control. If FOR UPDATE is specified with additional parameters, SAP Sybase IQ alters the processing of the statement to incorporate one of two concurrency control options as follows:

- **Pessimistic** – For all rows fetched in the cursor's result set, the database server acquires intent row locks to prevent the rows from being updated by any other transaction.
- **Optimistic** – The cursor type used by the database server is changed to a keyset-driven cursor (insensitive row membership, value-sensitive) so that the application can be informed when a row in the result has been modified or deleted by this, or any other transaction.

Pessimistic or optimistic concurrency is specified at the cursor level either through options with DECLARE CURSOR or FOR statements, or through the concurrency setting API for a specific programming interface. If a statement is updatable and the cursor does not specify a concurrency control mechanism, the statement's specification is used. The syntax is as follows:

- **FOR UPDATE BY LOCK** – The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until transaction COMMIT or ROLLBACK.

- **FOR UPDATE BY { VALUES | TIMESTAMP }** – The database server utilizes a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

Restricting updatable statements

FOR UPDATE (*column-list*) enforces the restriction that only named result set attributes can be modified in a subsequent UPDATE WHERE CURRENT OF statement.

Cursor operations that are canceled

You can cancel a request through an interface function. If you cancel a request that is carrying out a cursor operation, the position of the cursor is indeterminate. After canceling the request, you must locate the cursor by its absolute position, or close it.

Cursor types

This section describes mappings between SAP Sybase IQ cursors and the options available to you from the programming interfaces supported by SAP Sybase IQ.

Availability of cursors

Not all interfaces provide support for all types of cursors.

- ADO.NET provides only forward-only, read-only cursors.
- ADO/OLE DB and ODBC support all types of cursors.
- Embedded SQL™ supports all types of cursors.
- For JDBC:
 - The SQL Anywhere JDBC driver supports the JDBC 4.0 specification and permits the declaration of insensitive, sensitive, and forward-only asensitive cursors.
 - jConnect supports the declaration of insensitive, sensitive, and forward-only asensitive cursors in the same manner as the SQL Anywhere JDBC driver. However, the underlying implementation of jConnect only supports asensitive cursor semantics.
- Sybase Open Client supports only asensitive cursors. Also, a severe performance penalty results when using updatable, non-unique cursors.

Cursor properties

You request a cursor type, either explicitly or implicitly, from the programming interface. Different interface libraries offer different choices of cursor types. For example, JDBC and ODBC specify different cursor types.

Each cursor type is defined by several characteristics:

- **Uniqueness** – Declaring a cursor to be unique forces the query to return all the columns required to uniquely identify each row. Often this means returning all the columns in the

primary key. Any columns required but not specified are added to the result set. The default cursor type is non-unique.

- **Updatability** – A cursor declared as read-only cannot be used in a positioned update or delete operation. The default cursor type is updatable.
- **Scrollability** – You can declare cursors to behave different ways as you move through the result set. Some cursors can fetch only the current row or the following row. Others can move backward and forward through the result set.
- **Sensitivity** – Changes to the database may or may not be visible through a cursor.

These characteristics may have significant side effects on performance and on database server memory usage.

SAP Sybase IQ makes available cursors with a variety of mixes of these characteristics. When you request a cursor of a given type, SAP Sybase IQ tries to match those characteristics.

There are some occasions when not all characteristics can be supplied. For example, insensitive cursors in SAP Sybase IQ must be read-only. If your application requests an updatable insensitive cursor, a different cursor type (value-sensitive) is supplied instead.

Bookmarks and cursors

ODBC provides *bookmarks*, or values, used to identify rows in a cursor. SAP Sybase IQ supports bookmarks for value-sensitive and insensitive cursors. For example, the ODBC cursor types `SQL_CURSOR_STATIC` and `SQL_CURSOR_KEYSET_DRIVEN` support bookmarks while cursor types `SQL_CURSOR_DYNAMIC` and `SQL_CURSOR_FORWARD_ONLY` do not.

Block cursors

ODBC provides a cursor type called a block cursor. When you use a `BLOCK` cursor, you can use `SQLFetchScroll` or `SQLExtendedFetch` to fetch a block of rows, rather than a single row. Block cursors behave identically to embedded SQL `ARRAY` fetches.

SAP Sybase IQ Catalog Store Cursors

Any SAP Sybase IQ Catalog store cursor, once opened, has an associated result set. The cursor is kept open for a length of time. During that time, the result set associated with the cursor may be changed, either through the cursor itself or, subject to isolation level requirements, by other transactions. Some cursors permit changes to the underlying data to be visible, while others do not reflect these changes. A sensitivity to changes to the underlying data causes different cursor behavior, or *cursor sensitivity*.

The SAP Sybase IQ Catalog store provides cursors with a variety of sensitivity characteristics. This section describes what sensitivity is, and describes the sensitivity characteristics of cursors.

Membership, order, and value changes

Changes to the underlying data can affect the result set of a cursor in the following ways:

- **Membership** – The set of rows in the result set, as identified by their primary key values.
- **Order** – The order of the rows in the result set.
- **Value** – The values of the rows in the result set.

For example, consider the following simple table with employee information (EmployeeID is the primary key column):

EmployeeID	Surname
1	Whitney
2	Cobb
3	Chin

A cursor on the following query returns all results from the table in primary key order:

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

The membership of the result set could be changed by adding a new row or deleting a row. The values could be changed by changing one of the names in the table. The order could be changed by changing the primary key value of one of the employees.

Visible and invisible changes

Subject to isolation level requirements, the membership, order, and values of the result set of a cursor can be changed after the cursor is opened. Depending on the type of cursor in use, the result set as seen by the application may or may not change to reflect these changes.

Changes to the underlying data may be *visible* or *invisible* through the cursor. A visible change is a change that is reflected in the result set of the cursor. Changes to the underlying data that are not reflected in the result set seen by the cursor are invisible.

Catalog Store Cursor Sensitivity

SAP Sybase IQ cursors are classified by their sensitivity to changes in the underlying data. In other words, cursor sensitivity is defined by the changes that are visible.

- **Insensitive cursors** – The result set is fixed when the cursor is opened. No changes to the underlying data are visible.
- **Sensitive cursors** – The result set can change after the cursor is opened. All changes to the underlying data are visible.
- **Asensitive cursors** – Changes may be reflected in the membership, order, or values of the result set seen through the cursor, or may not be reflected at all.

- **Value-sensitive cursors** – Changes to the order or values of the underlying data are visible. The membership of the result set is fixed when the cursor is opened.

The differing requirements on cursors place different constraints on execution and therefore affect performance.

Cursor sensitivity example: A deleted row

This example uses a simple query to illustrate how different cursors respond to a row in the result set being deleted.

Consider the following sequence of events:

1. An application opens a cursor on the following query against the sample database.

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. The application fetches the first row through the cursor (102).
3. The application fetches the next row through the cursor (105).
4. A separate transaction deletes employee 102 (Whitney) and commits the change.

The results of cursor actions in this situation depend on the cursor sensitivity:

- **Insensitive cursors** – The DELETE is not reflected in either the membership or values of the results as seen through the cursor:

Action	Result
Fetch previous row	Returns the original copy of the row (102).
Fetch the first row (absolute fetch)	Returns the original copy of the row (102).
Fetch the second row (absolute fetch)	Returns the unchanged row (105).

- **Sensitive cursors** – The membership of the result set has changed so that row 105 is now the first row in the result set:

Action	Result
Fetch previous row	Returns Row Not Found. There is no previous row.

Action	Result
Fetch the first row (absolute fetch)	Returns row 105.
Fetch the second row (absolute fetch)	Returns row 160.

- **Value-sensitive cursors** – The membership of the result set is fixed, and so row 105 is still the second row of the result set. The DELETE is reflected in the values of the cursor, and creates an effective hole in the result set.

Action	Result
Fetch previous row	Returns No current row of cursor. There is a hole in the cursor where the first row used to be.
Fetch the first row (absolute fetch)	Returns No current row of cursor. There is a hole in the cursor where the first row used to be.
Fetch the second row (absolute fetch)	Returns row 105.

- **Asensitive cursors** – For changes, the membership and values of the result set are indeterminate. The response to a fetch of the previous row, the first row, or the second row depends on the particular optimization method for the query, whether that method involved the formation of a work table, and whether the row being fetched was prefetched from the client.

The benefit of asensitive cursors is that for many applications, sensitivity is unimportant. In particular, if you are using a forward-only, read-only cursor, no underlying changes are seen. Also, if you are running at a high isolation level, underlying changes are disallowed.

Cursor sensitivity example: An updated row

This example uses a simple query to illustrate how different cursor types respond to a row in the result set being updated in such a way that the order of the result set is changed.

Consider the following sequence of events:

1. An application opens a cursor on the following query against the sample database.

```
SELECT EmployeeID, Surname
FROM Employees;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault

EmployeeID	Surname
...	...

- The application fetches the first row through the cursor (102).
- The application fetches the next row through the cursor (105).
- A separate transaction updates the employee ID of employee 102 (Whitney) to 165 and commits the change.

The results of the cursor actions in this situation depend on the cursor sensitivity:

- Insensitive cursors** – The UPDATE is not reflected in either the membership or values of the results as seen through the cursor:

Action	Result
Fetch previous row	Returns the original copy of the row (102).
Fetch the first row (absolute fetch)	Returns the original copy of the row (102).
Fetch the second row (absolute fetch)	Returns the unchanged row (105).

- Sensitive cursors** – The membership of the result set has changed so that row 105 is now the first row in the result set:

Action	Result
Fetch previous row	Returns SQLCODE 100. The membership of the result set has changed so that 105 is now the first row. The cursor is moved to the position before the first row.
Fetch the first row (absolute fetch)	Returns row 105.
Fetch the second row (absolute fetch)	Returns row 160.

In addition, a fetch on a sensitive cursor returns a `SQL_ROW_UPDATED_WARNING` warning if the row has changed since the last reading. The warning is given only once. Subsequent fetches of the same row do not produce the warning.

Similarly, a positioned update or delete through the cursor on a row since it was last fetched returns the `SQL_ROW_UPDATED_SINCE_READ` error. An application must fetch the row again for an update or delete on a sensitive cursor to work.

An update to any column causes the warning/error, even if the column is not referenced by the cursor. For example, a cursor on a query returning Surname would report the update even if only the Salary column was modified.

- Value-sensitive cursors** – The membership of the result set is fixed, and so row 105 is still the second row of the result set. The UPDATE is reflected in the values of the cursor, and creates an effective "hole" in the result set.

Action	Result
Fetch previous row	Returns SQLCODE 100. The membership of the result set has changed so that 105 is now the first row: The cursor is positioned on the hole: it is before row 105.
Fetch the first row (absolute fetch)	Returns SQLCODE -197. The membership of the result set has changed so that 105 is now the first row: The cursor is positioned on the hole: it is before row 105.
Fetch the second row (absolute fetch)	Returns row 105.

- Asensitive cursors** – For changes, the membership and values of the result set are indeterminate. The response to a fetch of the previous row, the first row, or the second row depends on the particular optimization method for the query, whether that method involved the formation of a work table, and whether the row being fetched was prefetched from the client.

Note: Update warning and error conditions do not occur in bulk operations mode (-b database server option).

Catalog Store Insensitive Cursors

These cursors have insensitive membership, order, and values. No changes made after cursor open time are visible.

Insensitive cursors are used only for read-only cursor types.

Standards

Insensitive cursors correspond to the ISO/ANSI standard definition of insensitive cursors, and to ODBC static cursors.

Programming interfaces

Interface	Cursor type	Comment
ODBC, ADO/OLE DB	Static	If an updatable static cursor is requested, a value-sensitive cursor is used instead.
Embedded SQL	INSENSITIVE	
JDBC	INSENSITIVE	Insensitive semantics are only supported by the SQL Anywhere JDBC driver.
Open Client	Unsupported	

Description

Insensitive cursors always return rows that match the query's selection criteria, in the order specified by any ORDER BY clause.

The result set of an insensitive cursor is fully materialized as a work table when the cursor is opened. This has the following consequences:

- If the result set is very large, the disk space and memory requirements for managing the result set may be significant.
- No row is returned to the application before the entire result set is assembled as a work table. For complex queries, this may lead to a delay before the first row is returned to the application.
- Subsequent rows can be fetched directly from the work table, and so are returned quickly. The client library may prefetch several rows at a time, further improving performance.
- Insensitive cursors are not affected by ROLLBACK or ROLLBACK TO SAVEPOINT.

Catalog Store Sensitive Cursors

Sensitive cursors can be used for read-only or updatable cursor types.

These cursors have sensitive membership, order, and values.

Standards

Sensitive cursors correspond to the ISO/ANSI standard definition of sensitive cursors, and to ODBC dynamic cursors.

Programming interfaces

Interface	Cursor type	Comment
ODBC, ADO/OLE DB	Dynamic	
Embedded SQL	SENSITIVE	Also supplied in response to a request for a DYNAMIC SCROLL cursor when no work table is required and the pre-fetch option is set to Off.
JDBC	SENSITIVE	Sensitive cursors are fully supported by the SQL Anywhere JDBC driver.

Description

Prefetching is disabled for sensitive cursors. All changes are visible through the cursor, including changes through the cursor and from other transactions. Higher isolation levels may hide some changes made in other transactions because of locking.

Changes to cursor membership, order, and all column values are all visible. For example, if a sensitive cursor contains a join, and one of the values of one of the underlying tables is modified, then all result rows composed from that base row show the new value. Result set membership and order may change at each fetch.

Sensitive cursors always return rows that match the query's selection criteria, and are in the order specified by any `ORDER BY` clause. Updates may affect the membership, order, and values of the result set.

The requirements of sensitive cursors place restrictions on the implementation of sensitive cursors:

- Rows cannot be prefetched, as changes to the prefetched rows would not be visible through the cursor. This may impact performance.
- Sensitive cursors must be implemented without any work tables being constructed, as changes to those rows stored as work tables would not be visible through the cursor.
- The no work table limitation restricts the choice of join method by the optimizer and therefore may impact performance.
- For some queries, the optimizer is unable to construct a plan that does not include a work table that would make a cursor sensitive.

Work tables are commonly used for sorting and grouping intermediate results. A work table is not needed for sorting if the rows can be accessed through an index. It is not possible to state exactly which queries employ work tables, but the following queries do employ them:

- `UNION` queries, although `UNION ALL` queries do not necessarily use work tables.
- Statements with an `ORDER BY` clause, if there is no index on the `ORDER BY` column.
- Any query that is optimized using a hash join.
- Many queries involving `DISTINCT` or `GROUP BY` clauses.

In these cases, SAP Sybase IQ either returns an error to the application, or changes the cursor type to an asensitive cursor and returns a warning.

Catalog Store Asensitive Cursors

These cursors do not have well-defined sensitivity in their membership, order, or values. The flexibility that is allowed in the sensitivity permits asensitive cursors to be optimized for performance.

Asensitive cursors are used only for read-only cursor types.

Standards

Asensitive cursors correspond to the ISO/ANSI standard definition of asensitive cursors, and to ODBC cursors with unspecified sensitivity.

Programming interfaces

Interface	Cursor type
ODBC, ADO/OLE DB	Unspecified sensitivity
Embedded SQL	DYNAMIC SCROLL

Description

A request for an asensitive cursor places few restrictions on the methods SAP Sybase IQ can use to optimize the query and return rows to the application. For these reasons, asensitive cursors provide the best performance. In particular, the optimizer is free to employ any measure of materialization of intermediate results as work tables, and rows can be prefetched by the client.

SAP Sybase IQ makes no guarantees about the visibility of changes to base underlying rows. Some changes may be visible, others not. Membership and order may change at each fetch. In particular, updates to base rows may result in only some of the updated columns being reflected in the cursor's result.

Asensitive cursors do not guarantee to return rows that match the query's selection and order. The row membership is fixed at cursor open time, but subsequent changes to the underlying values are reflected in the results.

Asensitive cursors always return rows that matched the customer's WHERE and ORDER BY clauses at the time the cursor membership is established. If column values change after the cursor is opened, rows may be returned that no longer match WHERE and ORDER BY clauses.

Catalog Store Value-Sensitive Cursors

For value-sensitive cursors, membership is insensitive, and the order and value of the result set is sensitive.

Value-sensitive cursors can be used for read-only or updatable cursor types.

Standards

Value-sensitive cursors do not correspond to an ISO/ANSI standard definition. They correspond to ODBC keyset-driven cursors.

Programming interfaces

Interface	Cursor type	Comment
ODBC, ADO/OLE DB	Keyset-driven	
Embedded SQL	SCROLL	

Interface	Cursor type	Comment
JDBC	INSENSITIVE and CONCUR_UPDATABLE	With the SQL Anywhere JDBC driver, a request for an updatable INSENSITIVE cursor is answered with a value-sensitive cursor.
Open Client and jConnect	Not supported	

Description

If the application fetches a row composed of a base underlying row that has changed, then the application must be presented with the updated value, and the SQL_ROW_UPDATED status must be issued to the application. If the application attempts to fetch a row that was composed of a base underlying row that was deleted, a SQL_ROW_DELETED status must be issued to the application.

Changes to primary key values remove the row from the result set (treated as a delete, followed by an insert). A special case occurs when a row in the result set is deleted (either from cursor or outside) and a new row with the same key value is inserted. This will result in the new row replacing the old row where it appeared.

There is no guarantee that rows in the result set match the query's selection or order specification. Since row membership is fixed at open time, subsequent changes that make a row not match the WHERE clause or ORDER BY do not change a row's membership nor position.

All values are sensitive to changes made through the cursor. The sensitivity of membership to changes made through the cursor is controlled by the ODBC option SQL_STATIC_SENSITIVITY. If this option is on, then inserts through the cursor add the row to the cursor. Otherwise, they are not part of the result set. Deletes through the cursor remove the row from the result set, preventing a hole returning the SQL_ROW_DELETED status.

Value-sensitive cursors use a *key set table*. When the cursor is opened, SAP Sybase IQ populates a work table with identifying information for each row contributing to the result set. When scrolling through the result set, the key set table is used to identify the membership of the result set, but values are obtained, if necessary, from the underlying tables.

The fixed membership property of value-sensitive cursors allows your application to remember row positions within a cursor and be assured that these positions will not change.

- If a row was updated or may have been updated since the cursor was opened, SAP Sybase IQ returns a SQLE_ROW_UPDATED_WARNING when the row is fetched. The warning is generated only once: fetching the same row again does not produce the warning. An update to any column of the row causes the warning, even if the updated column is not referenced by the cursor. For example, a cursor on Surname and GivenName would report the update even if only the Birthdate column was modified. These update warning and

error conditions do not occur in bulk operations mode (-b database server option) when row locking is disabled.

- An attempt to execute a positioned update or delete on a row that has been modified since it was last fetched returns a `SQL_ROW_UPDATED_SINCE_READ` error and cancels the statement. An application must `FETCH` the row again before the `UPDATE` or `DELETE` is permitted.

An update to any column of the row causes the error, even if the updated column is not referenced by the cursor. The error does not occur in bulk operations mode.

- If a row has been deleted after the cursor is opened, either through the cursor or from another transaction, a *hole* is created in the cursor. The membership of the cursor is fixed, so a row position is reserved, but the `DELETE` operation is reflected in the changed value of the row. If you fetch the row at this hole, you receive a -197 `SQLCODE` error, indicating that there is no current row, and the cursor is left positioned on the hole. You can avoid holes by using sensitive cursors, as their membership changes along with the values.

Rows cannot be prefetched for value-sensitive cursors. This requirement may affect performance.

Inserting multiple rows

When inserting multiple rows through a value-sensitive cursor, the new rows appear at the end of the result set.

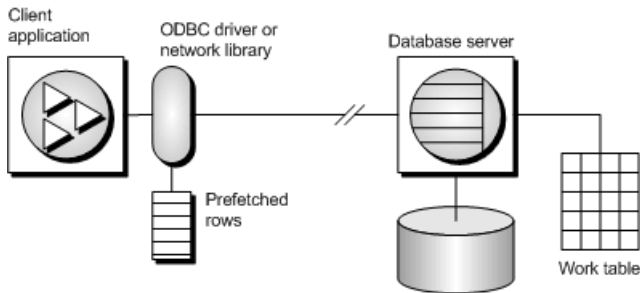
Catalog Store Cursor Sensitivity and Performance

There is a trade-off between performance and other cursor properties. In particular, making a cursor updatable places restrictions on the cursor query processing and delivery that constrain performance. Also, putting requirements on cursor sensitivity may constrain cursor performance.

To understand how the updatability and sensitivity of cursors affects performance, you need to understand how the results that are visible through a cursor are transmitted from the database to the client application.

In particular, results may be stored at two intermediate locations for performance reasons:

- **Work tables** – Either intermediate or final results may be stored as work tables. Value-sensitive cursors employ a work table of primary key values. Query characteristics may also lead the optimizer to use work tables in its chosen execution plan.
- **Prefetching** – The client side of the communication may retrieve rows into a buffer on the client side to avoid separate requests to the database server for each row.



Sensitivity and updatability limit the use of intermediate locations.

Prefetches

Prefetches and multiple-row fetches are different. Prefetches can be carried out without explicit instructions from the client application. Prefetching retrieves rows from the server into a buffer on the client side, but does not make those rows available to the client application until the application fetches the appropriate row.

By default, the SAP Sybase IQ client library prefetches multiple rows whenever an application fetches a single row. The SAP Sybase IQ client library stores the additional rows in a buffer.

Prefetching assists performance by cutting down on client/server round trips, and increases throughput by making many rows available without a separate request to the server for each row or block of rows.

Controlling prefetching from an application

- The prefetch option controls whether prefetching occurs. You can set the prefetch option to Always, Conditional, or Off for a single connection. By default, it is set to Conditional.
- In embedded SQL, you can control prefetching on a per-cursor basis when you open a cursor on an individual FETCH operation using the BLOCK clause.

The application can specify a maximum number of rows contained in a single fetch from the server by specifying the BLOCK clause. For example, if you are fetching and displaying 5 rows at a time, you could use BLOCK 5. Specifying BLOCK 0 fetches 1 record at a time and also causes a FETCH RELATIVE 0 to always fetch the row from the server again.

Although you can also turn off prefetch by setting a connection parameter on the application, it is more efficient to specify BLOCK 0 than to set the prefetch option to Off.

- Prefetch is disabled by default for value sensitive cursor types.
- In Open Client, you can control prefetching behavior using `ct_cursor` with `CS_CURSOR_ROWS` after the cursor is declared, but before it is opened.

Prefetch dynamically increases the number of prefetch rows when improvements in performance could be achieved. This includes cursors that meet the following conditions:

- They use one of the supported cursor types:
 - **ODBC and OLE DB** – FORWARD-ONLY and READ-ONLY (default) cursors
 - **Embedded SQL** – DYNAMIC SCROLL (default), NO SCROLL, and INSENSITIVE cursors
 - **ADO.NET** – all cursors
- They perform only FETCH NEXT operations (no absolute, relative, or backward fetching).
- The application does not change the host variable type between fetches and does not use a GET DATA statement to get column data in chunks (using one GET DATA statement to get the value is supported).

Lost updates

When using an updatable cursor, it is important to guard against lost updates. A lost update is a scenario in which two or more transactions update the same row, but neither transaction is aware of the modification made by the other transaction, and the second change overwrites the first modification. The following example illustrates this problem:

1. An application opens a cursor on the following query against the sample database.

```
SELECT ID, Quantity
FROM Products;
```

ID	Quantity
300	28
301	54
302	75
...	...

2. The application fetches the row with ID = 300 through the cursor.
3. A separate transaction updates the row using the following statement:


```
UPDATE Products
SET Quantity = Quantity - 10
WHERE ID = 300;
```
4. The application then updates the row through the cursor to a value of (Quantity - 5).
5. The correct final value for the row would be 13. If the cursor had prefetched the row, the new value of the row would be 23. The update from the separate transaction is lost.

In a database application, the potential for a lost update exists at any isolation level if changes are made to rows without verification of their values beforehand. At higher isolation levels (2 and 3), locking (read, intent, and write locks) can be used to ensure that changes to rows cannot be made by another transaction once the row has been read by the application. However, at isolation levels 0 and 1, the potential for lost updates is greater: at isolation level 0, read locks are not acquired to prevent subsequent changes to the data, and isolation level 1 only locks the current row. Lost updates cannot occur when using snapshot isolation since any attempt to

change an old value results in an update conflict. Also, the use of prefetching at isolation level 1 can also introduce the potential for lost updates, since the result set row that the application is positioned on, which is in the client's prefetch buffer, may not be the same as the current row that the server is positioned on in the cursor.

To prevent lost updates from occurring with cursors at isolation level 1, the database server supports three different concurrency control mechanisms that can be specified by an application:

1. The acquisition of intent row locks on each row in the cursor as it is fetched. Intent locks prevent other transactions from acquiring intent or write locks on the same row, preventing simultaneous updates. However, intent locks do not block read row locks, so they do not affect the concurrency of read-only statements.
2. The use of a value-sensitive cursor. Value-sensitive cursors can be used to track when an underlying row has changed, or has been deleted, so that the application can respond.
3. The use of `FETCH FOR UPDATE`, which acquires an intent row lock for that specific row.

How these alternatives are specified depends on the interface used by the application. For the first two alternatives that pertain to a `SELECT` statement:

- In ODBC, lost updates cannot occur because the application must specify a cursor concurrency parameter to the `SQLSetStmtAttr` function when declaring an updatable cursor. This parameter is one of `SQL_CONCUR_LOCK`, `SQL_CONCUR_VALUES`, `SQL_CONCUR_READ_ONLY`, or `SQL_CONCUR_TIMESTAMP`. For `SQL_CONCUR_LOCK`, the database server acquires row intent locks. For `SQL_CONCUR_VALUES` and `SQL_CONCUR_TIMESTAMP`, a value-sensitive cursor is used. `SQL_CONCUR_READ_ONLY` is used for read-only cursors, and is the default.
- In JDBC, the concurrency setting for a statement is similar to that of ODBC. The SQL Anywhere JDBC driver supports the JDBC concurrency values `RESULTSET_CONCUR_READ_ONLY` and `RESULTSET_CONCUR_UPDATABLE`. The first value corresponds to the ODBC concurrency setting `SQL_CONCUR_READ_ONLY` and specifies a read-only statement. The second value corresponds to the ODBC `SQL_CONCUR_LOCK` setting, so row intent locks are used to prevent lost updates. Value-sensitive cursors cannot be specified directly in the JDBC 4.0 specification.
- In jConnect, updatable cursors are supported at the API level, but the underlying implementation (using TDS) does not support updates through a cursor. Instead, jConnect sends a separate `UPDATE` statement to the database server to update the specific row. To avoid lost updates, the application must run at isolation level 2 or higher. Alternatively, the application can issue separate `UPDATE` statements from the cursor, but you must ensure that the `UPDATE` statement verifies that the row values have not been altered since the row was read by placing appropriate conditions in the `UPDATE` statement's `WHERE` clause.
- In embedded SQL, a concurrency specification can be set by including syntax within the `SELECT` statement itself, or in the cursor declaration. In the `SELECT` statement, the

syntax `SELECT...FOR UPDATE BY LOCK` causes the database server to acquire intent row locks on the result set.

Alternatively, `SELECT...FOR UPDATE BY [VALUES | TIMESTAMP]` causes the database server to change the cursor type to a value-sensitive cursor, so that if a specific row has been changed since the row was last read through the cursor, the application receives either a warning (`SQLE_ROW_UPDATED_WARNING`) on a `FETCH` statement, or an error (`SQLE_ROW_UPDATED_SINCE_READ`) on an `UPDATE WHERE CURRENT OF` statement. If the row was deleted, the application also receives an error (`SQLE_NO_CURRENT_ROW`).

`FETCH FOR UPDATE` functionality is also supported by the embedded SQL and ODBC interfaces, although the details differ depending on the API that is used.

In embedded SQL, the application uses `FETCH FOR UPDATE`, rather than `FETCH`, to cause an intent lock to be acquired on the row. In ODBC, the application uses the API call `SQLSetPos` with the operation argument `SQL_POSITION` or `SQL_REFRESH`, and the lock type argument `SQL_LOCK_EXCLUSIVE`, to acquire an intent lock on a row. In SAP Sybase IQ, these are long-term locks that are held until the transaction commits or rolls back.

Catalog Store Cursor Sensitivity and Isolation Levels

Both cursor sensitivity and isolation levels address the problem of concurrency control, but in different ways, and with different sets of tradeoffs.

By choosing an isolation level for a transaction (typically at the connection level), you determine the type and locks to place, and when, on rows in the database. Locks prevent other transactions from accessing or modifying rows in the database. In general, the greater the number of locks held, the lower the expected level of concurrency across concurrent transactions.

However, locks do not prevent updates from other portions of the same transaction from occurring. So, a single transaction that maintains multiple updatable cursors cannot rely on locking to prevent such problems as lost updates.

Snapshot isolation is intended to eliminate the need for read locks by ensuring that each transaction sees a consistent view of the database. The obvious advantage is that a consistent view of the database can be queried without relying on fully serializable transactions (isolation level 3), and the loss of concurrency that comes with using isolation level 3. However, snapshot isolation comes with a significant cost because copies of modified rows must be maintained to satisfy the requirements of both concurrent snapshot transactions already executing, and snapshot transactions that have yet to start. Because of this copy maintenance, the use of snapshot isolation may be inappropriate for heavy-update workloads.

Cursor sensitivity, however, determines which changes are visible (or not) to the cursor's result. Because cursor sensitivity is specified on a cursor basis, cursor sensitivity applies to both the effects of other transactions and to update activity of the same transaction, although these effects depend entirely on the cursor type specified. By setting cursor sensitivity, you are not directly determining when locks are placed on rows in the database. However, it is the

combination of cursor sensitivity and isolation level that controls the various concurrency scenarios that are possible with a particular application.

Requests for SAP Sybase IQ Catalog Store Cursors

When you request a cursor type from your client application, SAP Sybase IQ provides a cursor. SAP Sybase IQ cursors are defined, not by the type as specified in the programming interface, but by the sensitivity of the result set to changes in the underlying data. Depending on the cursor type you ask for, SAP Sybase IQ provides a cursor with behavior to match the type.

SAP Sybase IQ cursor sensitivity is set in response to the client cursor type request.

ADO.NET

Forward-only, read-only cursors are available by using `SACCommand.ExecuteReader`. The `SADDataAdapter` object uses a client-side result set instead of cursors.

ADO/OLE DB and ODBC

The following table illustrates the cursor sensitivity that is set in response to different ODBC scrollable cursor types.

ODBC scrollable cursor type	SAP Sybase IQ cursor
STATIC	Insensitive
KEYSET-DRIVEN	Value-sensitive
DYNAMIC	Sensitive
MIXED	Value-sensitive

A MIXED cursor is obtained by setting the cursor type to `SQL_CURSOR_KEYSET_DRIVEN`, and then specifying the number of rows in the keyset for a keyset-driven cursor using `SQL_ATTR_KEYSET_SIZE`. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside the keyset). The default keyset size is 0. It is an error if the keyset size is greater than 0 and less than the rowset size (`SQL_ATTR_ROW_ARRAY_SIZE`).

Exceptions

If a STATIC cursor is requested as updatable, a value-sensitive cursor is supplied instead and a warning is issued.

If a DYNAMIC or MIXED cursor is requested and the query cannot be executed without using work tables, a warning is issued and an asensitive cursor is supplied instead.

JDBC

The JDBC 4.0 specification supports three types of cursors: insensitive, sensitive, and forward-only asensitive. The SQL Anywhere JDBC driver is compliant with these JDBC

specifications and supports these different cursor types for a JDBC ResultSet object. However, there are cases when the database server cannot construct an access plan with the required semantics for a given cursor type. In these cases, the database server either returns an error or substitutes a different cursor type.

With jConnect, the underlying protocol (TDS) only supports forward-only, read-only asensitive cursors on the database server, even though jConnect supports the APIs for creating different types of cursors following the JDBC 2.0 specification. All jConnect cursors are asensitive because the TDS protocol buffers the statement's result set in blocks. These blocks of buffered results are scrolled when the application needs to scroll through an insensitive or sensitive cursor type that supports scrollability. If the application scrolls backward past the beginning of the cached result set, the statement is re-executed, which can result in data inconsistencies if the data has been altered between statement executions.

Embedded SQL

To request a cursor from an embedded SQL application, you specify the cursor type on the DECLARE statement. The following table illustrates the cursor sensitivity that is set in response to different requests:

Cursor type	SAP Sybase IQ cursor
NO SCROLL	Asensitive
DYNAMIC SCROLL	Asensitive
SCROLL	Value-sensitive
INSENSITIVE	Insensitive
SENSITIVE	Sensitive

Exceptions

If a DYNAMIC SCROLL or NO SCROLL cursor is requested as UPDATABLE, then a sensitive or value-sensitive cursor is supplied. It is not guaranteed which of the two is supplied. This uncertainty fits the definition of asensitive behavior.

If an INSENSITIVE cursor is requested as UPDATABLE, then a value-sensitive cursor is supplied.

If a DYNAMIC SCROLL cursor is requested, if the prefetch database option is set to Off, and if the query execution plan involves no work tables, then a sensitive cursor may be supplied. Again, this uncertainty fits the definition of asensitive behavior.

Open Client

As with jConnect, the underlying protocol (TDS) for Open Client only supports forward-only, read-only, asensitive cursors.

Result set descriptors

Some applications build SQL statements that cannot be completely specified in the application. Sometimes statements are dependent on a user response before the application knows exactly what information to retrieve, such as when a reporting application allows a user to select which columns to display.

In such a case, the application needs a method for retrieving information about both the nature of the *result set* and the contents of the result set. The information about the nature of the result set, called a *descriptor*, identifies the data structure, including the number and type of columns expected to be returned. Once the application has determined the nature of the result set, retrieving the contents is straightforward.

This *result set metadata* (information about the nature and content of the data) is manipulated using descriptors. Obtaining and managing the result set metadata is called *describing*.

Since cursors generally produce result sets, descriptors and cursors are closely linked, although some interfaces hide the use of descriptors from the user. Typically, statements needing descriptors are either SELECT statements or stored procedures that return result sets.

A sequence for using a descriptor with a cursor-based operation is as follows:

1. Allocate the descriptor. This may be done implicitly, although some interfaces allow explicit allocation as well.
2. Prepare the statement.
3. Describe the statement. If the statement is a stored procedure call or batch, and the result set is not defined by a result clause in the procedure definition, then the describe should occur after opening the cursor.
4. Declare and open a cursor for the statement (embedded SQL) or execute the statement.
5. Get the descriptor and modify the allocated area if necessary. This is often done implicitly.
6. Fetch and process the statement results.
7. Deallocate the descriptor.
8. Close the cursor.
9. Drop the statement. Some interfaces do this automatically.

Implementation notes

- In embedded SQL, a SQLDA (SQL Descriptor Area) structure holds the descriptor information.
- In ODBC, a descriptor handle allocated using SQLAllocHandle provides access to the fields of a descriptor. You can manipulate these fields using SQLSetDescRec, SQLSetDescField, SQLGetDescRec, and SQLGetDescField.
Alternatively, you can use SQLDescribeCol and SQLColAttributes to obtain column information.

- In Open Client, you can use `ct_dynamic` to prepare a statement and `ct_describe` to describe the result set of the statement. However, you can also use `ct_command` to send a SQL statement without preparing it first and use `ct_results` to handle the returned rows one by one. This is the more common way of operating in Open Client application development.
- In JDBC, the `java.sql.ResultSetMetaData` class provides information about result sets.
- You can also use descriptors for sending data to the database server (for example, with the INSERT statement); however, this is a different kind of descriptor than for result sets.

Transactions in applications

Transactions are sets of atomic SQL statements. Either all statements in the transaction are executed, or none. This section describes a few aspects of transactions in applications.

Autocommit and manual commit mode

Database programming interfaces can operate in either *manual commit* mode or *autocommit* mode.

- **Manual commit mode** – Operations are committed only when your application carries out an explicit commit operation or when the database server carries out an automatic commit, for example when executing an ALTER TABLE statement or other data definition statement. Manual commit mode is also sometimes called *chained mode*.

To use transactions in your application, including nested transactions and savepoints, you must operate in manual commit mode.

- **Autocommit mode** – Each statement is treated as a separate transaction. Autocommit mode is equivalent to appending a COMMIT statement to the end of each of your SQL statements. Autocommit mode is also sometimes called *unchained mode*.

Autocommit mode can affect the performance and behavior of your application. Do not use autocommit if your application requires transactional integrity.

How to control autocommit behavior

The way to control the commit behavior of your application depends on the programming interface you are using. The implementation of autocommit may be client-side or server-side, depending on the interface.

Control autocommit mode (ADO.NET)

By default, the ADO.NET provider operates in autocommit mode. To use explicit transactions, use the `SACConnection.BeginTransaction` method.

Control autocommit mode (OLE DB)

By default, the OLE DB provider operates in autocommit mode. To use explicit transactions, use the `ITransactionLocal::StartTransaction`, `ITransaction::Commit`, and `ITransaction::Abort` methods.

Control autocommit mode (ODBC)

By default, ODBC operates in autocommit mode. The way you turn off autocommit depends on whether you are using ODBC directly, or using an application development tool. If you are programming directly to the ODBC interface, set the `SQL_ATTR_AUTOCOMMIT` connection attribute.

Control autocommit mode (JDBC)

By default, JDBC operates in autocommit mode. To turn off autocommit, use the `setAutoCommit` method of the connection object:

```
conn.setAutoCommit( false );
```

Control autocommit mode (embedded SQL)

By default, embedded SQL applications operate in manual commit mode. To turn on autocommit, set the chained database option (a server-side option) to `Off` using a statement such as the following:

```
SET OPTION chained='Off';
```

Control autocommit mode (Open Client)

By default, a connection made through Open Client operates in autocommit mode. You can change this behavior by setting the chained database option (a server-side option) to `On` in your application using a statement such as the following:

```
SET OPTION chained='On';
```

Control autocommit mode (PHP)

By default, PHP operates in autocommit mode. To turn off autocommit, use the `sasql_set_option` function:

```
$result = sasql_set_option( $conn, "auto_commit", "Off" );
```

Control autocommit mode (on the server)

By default, the database server operates in manual commit mode. To turn on automatic commits, set the chained database option (a server-side option) to `Off` using a statement such as the following:

```
SET OPTION chained='Off';
```

If you are using an interface that controls commits on the client side, setting the chained database option (a server-side option) can impact performance and/or behavior of your application. Setting the server's chained mode is not recommended.

Autocommit implementation details

Autocommit mode has slightly different behavior depending on the interface and provider that you are using and how you control the autocommit behavior.

Autocommit mode can be implemented in one of two ways:

- **Client-side autocommit** – When an application uses autocommit, the client-library sends a COMMIT statement after each SQL statement executed.

ADO.NET, ADO/OLE DB, ODBC, PHP, and SQL Anywhere JDBC driver applications control commit behavior from the client side.

- **Server-side autocommit** – When an application turns off chained mode, the database server commits the results of each SQL statement. For the Sybase jConnect JDBC driver, this behavior is controlled by the chained database option.

Embedded SQL, the jConnect driver, and Open Client applications manipulate server-side commit behavior (for example, they set the chained option).

For compound statements such as stored procedures or triggers there is a difference between client-side and server-side autocommit. From the client side, a stored procedure is a single statement, and so autocommit sends a single commit statement after the whole procedure is executed. From the database server perspective, the stored procedure may be composed of many SQL statements, and so server-side autocommit commits the results of each SQL statement within the procedure.

Note: Do not mix client-side and server-side implementations. You should not combine the setting of the chained option with the setting of the autocommit option in your SAP Sybase IQ ADO.NET, OLE DB, ODBC, PHP, or JDBC application.

Isolation level settings

You can set the isolation level of a current connection using the `isolation_level` database option.

Some interfaces, such as ODBC, allow you to set the isolation level for a connection at connection time. You can reset this level later using the `isolation_level` database option.

You can override any temporary or public settings for the `isolation_level` database option within individual INSERT, UPDATE, DELETE, SELECT, and UNION statements by including an OPTION clause in the statement.

Cursors and transactions

In general, a cursor closes when a COMMIT is performed. There are two exceptions to this behavior.

- The `close_on_endtrans` database option is set to Off.
- A cursor is opened WITH HOLD, which is the default with Open Client and JDBC.

If either of these two cases is true, the cursor remains open on a COMMIT.

ROLLBACK and cursors

If a transaction rolls back, then cursors close except for those cursors opened WITH HOLD. However, don't rely on the contents of any cursor after a rollback.

The draft ISO SQL3 standard states that on a rollback, all cursors (even those cursors opened WITH HOLD) should close. You can obtain this behavior by setting the `ansi_close_cursors_on_rollback` option to On.

Savepoints

If a transaction rolls back to a savepoint, and if the `ansi_close_cursors_on_rollback` option is On, then all cursors (even those cursors opened WITH HOLD) opened after the SAVEPOINT close.

Cursors and isolation levels

You can change the isolation level of a connection during a transaction using the SET OPTION statement to alter the `isolation_level` option. However, this change does not affect open cursors.

A snapshot of all rows committed at the snapshot start time is visible when the WITH HOLD clause is used with the snapshot, statement-snapshot, and readonly-statement-snapshot isolation levels. Also visible are all modifications completed by the current connection since the start of the transaction within which the cursor was open.

.NET Application Programming

This section describes how to use SAP Sybase IQ with .NET, and includes the API for the SAP Sybase IQ .NET Data Provider.

SAP Sybase IQ .NET Data Provider

This section describes .NET support, including tips on using the SAP Sybase IQ .NET Data Provider in Visual Studio projects, connecting to databases, fetching, inserting, updating and deleting rows from database tables, calling stored procedures, using transactions, and basic error handling.

SAP Sybase IQ .NET Support

ADO.NET is the latest data access API from Microsoft in the line of ODBC, OLE DB, and ADO. It is the preferred data access component for the Microsoft .NET Framework and allows you to access relational database systems.

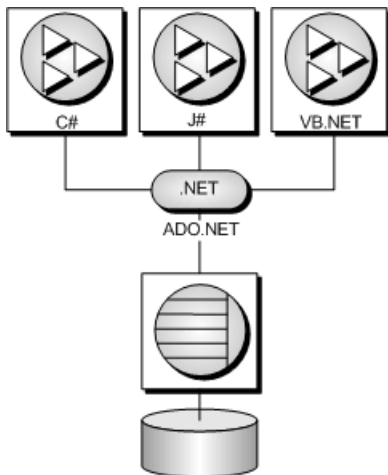
The SAP Sybase IQ .NET Data Provider implements the `iAnywhere.Data.SQLAnywhere` namespace and allows you to write programs in any of the .NET supported languages, such as C# and Visual Basic .NET, and access data from SAP Sybase IQ databases.

For general information about .NET data access, see the Microsoft ".NET Data Access Architecture Guide" at <http://msdn.microsoft.com/en-us/library/Ee817654%28pandp.10%29.aspx>.

ADO.NET applications

You can develop Internet and intranet applications using object-oriented languages, and then connect these applications to SAP Sybase IQ using the ADO.NET data provider.

Combine this provider with built-in XML and web services features, .NET scripting capability for MobiLink™ synchronization, and an UltraLite.NET™ component for development of handheld database applications, and SAP Sybase IQ can integrate with the .NET Framework.



SAP Sybase IQ .NET Data Provider Features

SAP Sybase IQ supports the Microsoft .NET Framework versions 2.0, 3.0, 3.5, 4.0, and 4.5 through three distinct namespaces.

- **iAnywhere.Data.SQLAnywhere** – The ADO.NET object model is an all-purpose data access model. ADO.NET components were designed to factor data access from data manipulation. There are two central components of ADO.NET that do this: the DataSet, and the .NET Framework data provider, which is a set of components including the Connection, Command, DataReader, and DataAdapter objects. SAP Sybase IQ includes a .NET Entity Framework Data Provider that communicates directly with an SAP Sybase IQ database server without adding the overhead of OLE DB or ODBC. The SAP Sybase IQ .NET Data provider is represented in the .NET namespace as `iAnywhere.Data.SQLAnywhere`.

The Microsoft .NET Compact Framework is the smart device development framework for Microsoft .NET. The SAP Sybase IQ .NET Compact Framework Data Provider supports devices running Windows Mobile. Compact Framework 2.0 and 3.5 are supported.

The SAP Sybase IQ .NET Data Provider namespace is described in this document.

To read more about how to access data stored inside an SAP Sybase IQ database using the ADO.NET object model, in particular via the Language Integrated Query (LINQ) to Entities methodology, see the *SQL Anywhere and the ADO.NET Entity Framework* white paper at www.sybase.com/detail?id=1060541.

- **System.Data.OleDb** – This namespace supports OLE DB data sources. This namespace is an intrinsic part of the Microsoft .NET Framework. You can use `System.Data.OleDb` together with the SQL Anywhere OLE DB provider, `SAOLEDB`, to access SAP Sybase IQ databases.

- **System.Data.Odbc** – This namespace supports ODBC data sources. This namespace is an intrinsic part of the Microsoft .NET Framework. You can use System.Data.Odbc together with the SQL Anywhere ODBC driver to access SAP Sybase IQ databases.

There are some key benefits to using the SAP Sybase IQ .NET Data Provider:

- In the .NET environment, the SAP Sybase IQ .NET Data Provider provides native access to an SAP Sybase IQ database. Unlike the other supported providers, it communicates directly with an SAP Sybase IQ server and does not require bridge technology.
- As a result, the SAP Sybase IQ .NET Data Provider is faster than the OLE DB and ODBC Data Providers. It is the recommended Data Provider for accessing SAP Sybase IQ databases.

.NET Sample Projects

There are several sample projects included with the SAP Sybase IQ .NET Data Provider:

- **LinqSample** – A .NET Framework sample project for Windows that demonstrates language-integrated query, set, and transform operations using the SAP Sybase IQ .NET Data Provider and C#.
- **SimpleWin32** – A .NET Framework sample project for Windows that demonstrates a simple listbox that is filled with the names from the Employees table when you click **Connect**.
- **SimpleXML** – A .NET Framework sample project for Windows that demonstrates how to obtain XML data from SAP Sybase IQ via ADO.NET. Samples for C#, Visual Basic, and Visual C++ are provided.
- **SimpleViewer** – A .NET Framework sample project for Windows.
- **TableViewer** – A .NET Framework sample project for Windows that allows you to enter and execute SQL statements.

Using the .NET Data Provider in a Visual Studio Project

Use the SAP Sybase IQ .NET Data Provider to develop .NET applications with Visual Studio by including both a reference to the SAP Sybase IQ .NET Data Provider, and a line in your source code referencing the SAP Sybase IQ .NET Data Provider classes.

Prerequisites

There are no prerequisites for this task.

Task

1. Start Visual Studio and open your project.
2. In the **Solution Explorer** window, right-click **References** and click **Add Reference**.

The reference indicates which provider to include and locates the code for the SAP Sybase IQ .NET Data Provider.

3. Click the **.NET** tab, and scroll through the list to locate any of the following:

- iAnywhere.Data.SQLAnywhere for .NET 2
- iAnywhere.Data.SQLAnywhere for .NET 3.5
- iAnywhere.Data.SQLAnywhere for .NET 4

4. Click the desired provider and then click **OK**.

The provider is added to the **References** folder in the **Solution Explorer** window of your project.

5. Specify a directive to your source code to assist with the use of the SAP Sybase IQ .NET Data Provider namespace and the defined types.

Add the following line to your project:

- If you are using C#, add the following line to the list of `using` directives at the beginning of your source code:

```
using iAnywhere.Data.SQLAnywhere;
```

- If you are using Visual Basic, add the following line at the beginning of source code:

```
Imports iAnywhere.Data.SQLAnywhere
```

The SAP Sybase IQ .NET Data Provider is set up for use with your .NET application.

.NET Database Connection Examples

To connect to a database, an `SACConnection` object must be created. The connection string can be specified when creating the object or it can be established later by setting the `ConnectionString` property.

A well-designed application should handle any errors that occur when attempting to connect to a database.

A connection to the database is created when the connection is opened and released when the connection is closed.

C# SACConnection example

The following C# code creates a button click handler that opens a connection to the SAP Sybase IQ sample database and then closes it. An exception handler is included.

```
private void button1_Click(object sender, EventArgs e)
{
    SACConnection conn = new SACConnection("Data Source=Sybase IQ
Demo");
    try
    {
        conn.Open();

        conn.Close();
    }
}
```

```

catch (SAException ex)
{
    MessageBox.Show(ex.Errors[0].Source + " : " +
        ex.Errors[0].Message + " (" +
        ex.Errors[0].NativeError.ToString() + ")",
        "Failed to connect");
}
}

```

Visual Basic SAConnection example

The following Visual Basic code creates a button click handler that opens a connection to the SAP Sybase IQ sample database and then closes it. An exception handler is included.

```

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
    Try
        conn.Open()

        conn.Close()
    Catch ex As SAException
        MessageBox.Show(ex.Errors(0).Source & " : " & _
            ex.Errors(0).Message & " (" & _
            ex.Errors(0).NativeError.ToString() & ")", _
            "Failed to connect")
    End Try
End Sub

```

Connection Pooling

The SAP Sybase IQ .NET Data Provider supports native .NET connection pooling. Connection pooling allows your application to reuse existing connections by saving the connection handle to a pool so it can be reused, rather than repeatedly creating a new connection to the database. Connection pooling is enabled by default.

Connection pooling is enabled and disabled using the `Pooling` option. The maximum pool size is set in your connection string using the `Max Pool Size` option. The minimum or initial pool size is set in your connection string using the `Min Pool Size` option. The default maximum pool size is 100, while the default minimum pool size is 0.

```
"Data Source=Sybase IQ Demo;Pooling=true;Max Pool Size=50;Min Pool Size=5"
```

When your application first attempts to connect to the database, it checks the pool for an existing connection that uses the same connection parameters you have specified. If a matching connection is found, that connection is used. Otherwise, a new connection is used. When you disconnect, the connection is returned to the pool so that it can be reused.

The SAP Sybase IQ database server also supports connection pooling. This feature is controlled using the `ConnectionPool (CPOOL)` connection parameter. However, the SAP Sybase IQ .NET Data Provider does not use this server feature and disables it (`CPOOL=NO`). All connection pooling is done in the .NET client application instead (client-side connection pooling).

Connection State

Once your application has established a connection to the database, you can check the connection state to ensure that the connection is still open before communicating a request to the database server. If a connection is closed, you can return an appropriate message to the user and/or attempt to reopen the connection.

The `SACConnection` class has a `State` property that can be used to check the state of the connection. Possible state values are `ConnectionState.Open` and `ConnectionState.Closed`.

The following code checks whether the `SACConnection` object has been initialized, and if it has, it checks that the connection is open. A message is returned to the user if the connection is not open.

```
if ( conn == null || conn.State != ConnectionState.Open )
{
    MessageBox.Show( "Connect to a database first", "Not
connected" );
    return;
}
```

Data Access and Manipulation

With the SAP Sybase IQ .NET Data Provider, there are two ways you can access data:

- **SACCommand object** – The `SACCommand` object is the recommended way of accessing and manipulating data in .NET.

The `SACCommand` object allows you to execute SQL statements that retrieve or modify data directly from the database. Using the `SACCommand` object, you can issue SQL statements and call stored procedures directly against the database.

Within an `SACCommand` object, an `SADataReader` is used to return read-only result sets from a query or stored procedure. The `SADataReader` returns only one row at a time, but this does not degrade performance because the SAP Sybase IQ client-side libraries use prefetch buffering to prefetch several rows at a time.

Using the `SACCommand` object allows you to group your changes into transactions rather than operating in autocommit mode. When you use the `SATransaction` object, locks are placed on the rows so that other users cannot modify them.

- **SADDataAdapter object** – The `SADDataAdapter` object retrieves the entire result set into a `DataSet`. A `DataSet` is a disconnected store for data that is retrieved from a database. You can then edit the data in the `DataSet` and when you are finished, the `SADDataAdapter` object updates the database with the changes made to the `DataSet`. When you use the `SADDataAdapter`, there is no way to prevent other users from modifying the rows in your `DataSet`. You need to include logic within your application to resolve any conflicts that may occur.

There is no performance impact from using the `SADataReader` within an `SACCommand` object to fetch rows from the database rather than the `SADDataAdapter` object.

SACommand: Fetch Data Using ExecuteReader and ExecuteScalar

The SACommand object allows you to execute a SQL statement or call a stored procedure against an SAP Sybase IQ database. You can use any of the following methods to retrieve data from the database:

- **ExecuteReader** – Issues a SQL query that returns a result set. This method uses a forward-only, read-only cursor. You can loop quickly through the rows of the result set in one direction.
- **ExecuteScalar** – Issues a SQL query that returns a single value. This can be the first column in the first row of the result set, or a SQL statement that returns an aggregate value such as COUNT or AVG. This method uses a forward-only, read-only cursor.

When using the SACommand object, you can use the SADATAReader to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

When using the SADATAReader, there are several Get methods available that you can use to return the results in the specified data type.

C# ExecuteReader Example

The following C# code opens a connection to the SAP Sybase IQ sample database and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();
SACommand cmd = new SACommand("SELECT Surname FROM Employees", conn);
SADATAReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
while (reader.Read())
{
    listEmployees.Items.Add(reader.GetString(0));
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

Visual Basic ExecuteReader Example

The following Visual Basic code opens a connection to the SAP Sybase IQ sample database and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
Dim cmd As New SACommand("SELECT Surname FROM Employees", conn)
Dim reader As SADATAReader
conn.Open()
reader = cmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (reader.Read())
    ListEmployees.Items.Add(reader.GetString(0))
Loop
```

```
ListEmployees.EndUpdate()  
conn.Close()
```

C# ExecuteScalar Example

The following C# code opens a connection to the SAP Sybase IQ sample database and uses the ExecuteScalar method to obtain a count of the number of male employees in the Employees table:

```
SACConnection conn = new SACConnection("Data Source=Sybase IQ Demo");  
conn.Open();  
SACCommand cmd = new SACCommand(  
    "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'", conn);  
int count = (int) cmd.ExecuteScalar();  
textBox1.Text = count.ToString();  
conn.Close();
```

SACCommand: Fetch Result Set Schema Using GetSchemaTable

You can obtain schema information about columns in a result set.

The GetSchemaTable method of the SADataReader class obtains information about the current result set. The GetSchemaTable method returns the standard .NET DataTable object, which provides information about all the columns in the result set, including column properties.

C# Schema Information Example

The following example obtains information about a result set using the GetSchemaTable method and binds the DataTable object to the datagrid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=Sybase IQ Demo" );  
conn.Open();  
SACCommand cmd = new SACCommand("SELECT * FROM Employees", conn);  
SADataReader reader = cmd.ExecuteReader();  
DataTable schema = reader.GetSchemaTable();  
reader.Close();  
conn.Close();  
dataGridView1.DataSource = schema;
```

SACCommand: Insert, Delete, and Update Rows Using ExecuteNonQuery

To perform an insert, update, or delete with the SACCommand object, use the ExecuteNonQuery function. The ExecuteNonQuery function issues a query (SQL statement or stored procedure) that does not return a result set.

You can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. You must be connected to a database to use the SACCommand object.

To set the isolation level for a SQL statement, you must use the SACCommand object as part of an SATransaction object. When you modify data without an SATransaction object, the provider operates in autocommit mode and any changes that you make are applied immediately.

C# ExecuteNonQuery DELETE and INSERT Example

The following example opens a connection to the SAP Sybase IQ sample database and uses the `ExecuteNonQuery` method to remove all departments whose ID is greater than or equal to 600 and then add two new rows to the Departments table. It displays the updated table in a datagrid.

```

SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();

SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600",
    conn);
deleteCmd.ExecuteNonQuery();

SACommand insertCmd = new SACommand(
    "INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES ( ?, ? )",
    conn );
SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
insertCmd.Parameters.Add( parm );
parm = new SAParameter();
parm.SADbType = SADbType.Char;
insertCmd.Parameters.Add( parm );

insertCmd.Parameters[0].Value = 600;
insertCmd.Parameters[1].Value = "Eastern Sales";
int recordsAffected = insertCmd.ExecuteNonQuery();

insertCmd.Parameters[0].Value = 700;
insertCmd.Parameters[1].Value = "Western Sales";
recordsAffected = insertCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();

```

C# ExecuteNonQuery UPDATE Example

The following example opens a connection to the SAP Sybase IQ sample database and uses the `ExecuteNonQuery` method to update the `DepartmentName` column to "Engineering" in all

rows of the Departments table where the DepartmentID is 100. It displays the updated table in a datagrid.

```
SACConnection conn = new SACConnection("Data Source=Sybase IQ Demo");
conn.Open();

SACCommand updateCmd = new SACCommand(
    "UPDATE Departments SET DepartmentName = 'Engineering' " +
    "WHERE DepartmentID = 100", conn );
int recordsAffected = updateCmd.ExecuteNonQuery();

SACCommand selectCmd = new SACCommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACCommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

SACCommand: Retrieve Primary Key Values for Newly Inserted Rows

If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain the primary key values generated by the data source.

C# SACCommand Primary Key Example

The following example shows how to obtain the primary key that is generated for a newly inserted row. The example uses an SACCommand object to call a SQL stored procedure and an SAParameter object to retrieve the primary key that it returns. For demonstration purposes, the example creates a sample table (adodotnet_primarykey) and the stored procedure (sp_adodotnet_primarykey) that will be used to insert rows and return primary key values.

```
SACConnection conn = new SACConnection( "Data Source=Sybase IQ Demo" );
conn.Open();

SACCommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ("
+
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE
```



```

sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;

SqlParameter parmId = new SqlParameter();
parmId.SDbType = SDbType.Integer;
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add(parmId);

SqlParameter parmName = new SqlParameter();
parmName.SDbType = SDbType.Char;
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add(parmName);

parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id1);

parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id2);

parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id3);

parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id4);

cmd.CommandText = "SELECT * FROM adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
SADataReader dr = cmd.ExecuteReader();
conn.Close();
dataGridView1.DataSource = dr;

```

SDataAdapter: Overview

The SDataAdapter retrieves a result set into a DataTable. A DataSet is a collection of tables (DataTables) and the relationships and constraints between those tables. The DataSet is built

into the .NET Framework, and is independent of the Data Provider used to connect to your database.

When you use the `SDataAdapter`, you must be connected to the database to fill a `DataTable` and to update the database with changes made to the `DataTable`. However, once the `DataTable` is filled, you can modify the `DataTable` while disconnected from the database.

If you do not want to apply your changes to the database right away, you can write the `DataSet`, including the data and/or the schema, to an XML file using the `WriteXml` method. Then, you can apply the changes at a later time by loading a `DataSet` with the `ReadXml` method. The following shows two examples.

```
ds.WriteXml("Employees.xml");  
ds.WriteXml("EmployeesWithSchema.xml", XmlWriteMode.WriteSchema);
```

For more information, see the .NET Framework documentation for `WriteXml` and `ReadXml`.

When you call the `Update` method to apply changes from the `DataSet` to the database, the `SDataAdapter` analyzes the changes that have been made and then invokes the appropriate statements, `INSERT`, `UPDATE`, or `DELETE`, as necessary. When you use the `DataSet`, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. If another user has a lock on the row you are trying to update, an exception is thrown.

Warning! Any changes you make to the `DataSet` are made while you are disconnected. Your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that may occur when changes from the `DataSet` are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

Resolving Conflicts When Using the SDataAdapter

When you use the `SDataAdapter`, no locks are placed on the rows in the database. This means there is the potential for conflicts to arise when you apply changes from the `DataSet` to the database. Your application should include logic to resolve or log conflicts that arise.

Some of the conflicts that your application logic should address include:

- **Unique primary keys** – If two users insert new rows into a table, each row must have a unique primary key. For tables with `AUTOINCREMENT` primary keys, the values in the `DataSet` may become out of sync with the values in the data source.
- **Updates made to the same value** – If two users modify the same value, your application should include logic to determine which value is correct.
- **Schema changes** – If a user modifies the schema of a table you have updated in the `DataSet`, the update will fail when you apply the changes to the database.
- **Data concurrency** – Concurrent applications should see a consistent set of data. The `SDataAdapter` does not place a lock on rows that it fetches, so another user can update a value in the database once you have retrieved the `DataSet` and are working offline.

Many of these potential problems can be avoided by using the `SACommand`, `SDataReader`, and `SATransaction` objects to apply changes to the database. The `SATransaction` object is

recommended because it allows you to set the isolation level for the transaction and it places locks on the rows so that other users cannot modify them.

To simplify the process of conflict resolution, you can design your INSERT, UPDATE, or DELETE statement to be a stored procedure call. By including INSERT, UPDATE, and DELETE statements in stored procedures, you can catch the error if the operation fails. In addition to the statement, you can add error handling logic to the stored procedure so that if the operation fails the appropriate action is taken, such as recording the error to a log file, or trying the operation again.

SADataAdapter: Fetch Data into a DataTable Using Fill

The SADataAdapter allows you to view a result set by using the Fill method to fill a DataTable with the results from a query and then binding the DataTable to a display grid.

When setting up an SADataAdapter, you can specify a SQL statement that returns a result set. When Fill is called to populate a DataTable, all the rows are fetched in one operation using a forward-only, read-only cursor. Once all the rows in the result set have been read, the cursor is closed. Changes made to the rows in a DataTable can be reflected to the database using the Update method.

You can use the SADataAdapter object to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

Warning! Any changes you make to a DataTable are made independently of the original database table. Your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that may occur when changes from the DataTable are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

C# SADataAdapter Fill Example Using a DataTable

The following example shows how to fill a DataTable using the SADataAdapter. It creates a new DataTable object named Results and a new SADataAdapter object. The SADataAdapter Fill method is used to fill the DataTable with the results of the query. The DataTable is then bound to the grid on the screen.

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
DataTable dt = new DataTable("Results");
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

C# SADataAdapter Fill Example Using a DataSet

The following example shows how to fill a DataTable using the SADataAdapter. It creates a new DataSet object and a new SADataAdapter object. The SADataAdapter Fill method is

used to create a `DataTable` table named `Results` in the `DataSet` and then fill it with the results of the query. The `Results DataTable` is then bound to the grid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
DataSet ds = new DataSet();
SADDataAdapter da = new SADDataAdapter("SELECT * FROM Employees",
conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

SADDataAdapter: Format a DataTable Using FillSchema

The `SADDataAdapter` allows you to configure the schema of a `DataTable` to match that of a specific query using the `FillSchema` method. The attributes of the columns in the `DataTable` will match those of the `SelectCommand` of the `SADDataAdapter` object. Unlike the `Fill` method, no rows are stored in the `DataTable`.

C# SADDataAdapter FillSchema Example Using a DataTable

The following example shows how to use the `FillSchema` method to set up a new `DataTable` object with the same schema as a result set. The `Additions DataTable` is then bound to the grid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SADDataAdapter da = new SADDataAdapter("SELECT * FROM Employees",
conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = dt;
```

C# SADDataAdapter FillSchema Example Using a DataSet

The following example shows how to use the `FillSchema` method to set up a new `DataTable` object with the same schema as a result set. The `DataTable` is added to the `DataSet` using the `Merge` method. The `Additions DataTable` is then bound to the grid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SADDataAdapter da = new SADDataAdapter("SELECT * FROM Employees",
conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
DataSet ds = new DataSet();
ds.Merge(dt);
conn.Close();
dataGridView1.DataSource = ds.Tables["Additions"];
```

SDataAdapter: Insert Rows using Update

An example showing how to use the Update method of SDataAdapter to add rows to a table.

C# SDataAdapter Insert Example

The example fetches the Departments table into a DataTable using the SelectCommand property and the Fill method of the SDataAdapter. It then adds two new rows to the DataTable and updates the Departments table from the DataTable using the InsertCommand property and the Update method of the SDataAdapter.

```

SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.InsertCommand = new SACommand(
    "INSERT INTO Departments( DepartmentID, DepartmentName ) " +
    "VALUES( ?, ? )", conn );
da.InsertCommand.UpdatedRowSource =
    UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

DataRow row1 = dataTable.NewRow();
row1[0] = 600;
row1[1] = "Eastern Sales";
dataTable.Rows.Add( row1 );

DataRow row2 = dataTable.NewRow();
row2[0] = 700;
row2[1] = "Western Sales";
dataTable.Rows.Add( row2 );

rowCount = da.Update( dataTable );

```

```
dataTable.Clear();  
rowCount = da.Fill( dataTable );  
conn.Close();  
dataGridView1.DataSource = dataTable;
```

SADaAdapter: Delete Rows Using Update

An example showing how to use the Update method of SADaAdapter to delete rows from a table.

C# SADaAdapter Delete Example

The example adds two new rows to the Departments table and then fetches this table into a DataTable using the SelectCommand property and the Fill method of the SADaAdapter. It then deletes some rows from the DataTable and updates the Departments table from the DataTable using the DeleteCommand property and the Update method of the SADaAdapter.

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );  
conn.Open();  
SACommand prepCmd = new SACommand("", conn);  
prepCmd.CommandText =  
    "DELETE FROM Departments WHERE DepartmentID >= 600";  
prepCmd.ExecuteNonQuery();  
prepCmd.CommandText =  
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";  
prepCmd.ExecuteNonQuery();  
prepCmd.CommandText =  
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";  
prepCmd.ExecuteNonQuery();  
  
SADaAdapter da = new SADaAdapter();  
da.MissingMappingAction = MissingMappingAction.Passthrough;  
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;  
da.SelectCommand = new SACommand(  
    "SELECT * FROM Departments", conn);  
da.DeleteCommand = new SACommand(  
    "DELETE FROM Departments WHERE DepartmentID = ?",  
    conn);  
da.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;  
  
SAParameter parm = new SAParameter();  
parm.SADbType = SADbType.Integer;  
parm.SourceColumn = "DepartmentID";  
parm.SourceVersion = DataRowVersion.Original;  
da.DeleteCommand.Parameters.Add(parm);  
  
DataTable dataTable = new DataTable("Departments");  
int rowCount = da.Fill(dataTable);  
  
foreach (DataRow row in dataTable.Rows)  
{  
    if (Int32.Parse(row[0].ToString()) > 500)  
    {  
        row.Delete();  
    }  
}
```

```

rowCount = da.Update(dataTable);

dataTable.Clear();
rowCount = da.Fill(dataTable);
conn.Close();
dataGridView1.DataSource = dataTable;

```

SDataAdapter: Update Rows using Update

An example showing how to use the Update method of SDataAdapter to update rows in a table.

C# SDataAdapter Update Example

The example adds two new rows to the Departments table and then fetches this table into a DataTable using the SelectCommand property and the Fill method of the SDataAdapter. It then modifies some values in the DataTable and updates the Departments table from the DataTable using the UpdateCommand property and the Update method of the SDataAdapter.

```

SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.UpdateCommand = new SACommand(
    "UPDATE Departments SET DepartmentName = ? " +
    "WHERE DepartmentID = ?",
    conn );
da.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.UpdateCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.UpdateCommand.Parameters.Add( parm );

```

```
DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

foreach ( DataRow row in dataTable.Rows )
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row[1] = (string)row[1] + "_Updated";
    }
}
rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

SDataAdapter: Retrieve Primary Key Values for Newly Inserted Rows

If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain the primary key values generated by the data source.

C# SDataAdapter primary key example

The following example shows how to obtain the primary key that is generated for a newly inserted row. The example uses an SDataAdapter object to call a SQL stored procedure and an SAParameter object to retrieve the primary key that it returns. For demonstration purposes, the example creates a sample table (adodotnet_primarykey) and the stored procedure (sp_adodotnet_primarykey) that will be used to insert rows and return primary key values.

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ("
+
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE
sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();
```



```

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

da.SelectCommand = new SACommand(
    "SELECT * FROM adodotnet_primarykey", conn);

da.InsertCommand = new SACommand(
    "sp_adodotnet_primarykey", conn);
da.InsertCommand.CommandType = CommandType.StoredProcedure;
da.InsertCommand.UpdatedRowSource =
UpdateRowSource.OutputParameters;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "ID";
parmId.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "Name";
parmName.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmName);

DataTable dataTable = new DataTable("Departments");
da.FillSchema(dataTable, SchemaType.Source);

DataRow row = dataTable.NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataTable.Rows.Add(row);

DataSet ds = new DataSet();
ds.Merge(dataTable);
da.Update(ds, "Departments");

conn.Close();
dataGridView1.DataSource = ds.Tables["Departments"];

```

BLOBs

When fetching long string values or binary data, there are methods that you can use to fetch the data in pieces. For binary data, use the `GetBytes` method, and for string data, use the `GetChars` method. Otherwise, BLOB data is treated in the same manner as any other data you fetch from the database.

C# GetChars BLOB example

The following example reads three columns from a result set. The first two columns are integers, while the third column is a LONG VARCHAR. The length of the third column is computed by reading this column with the `GetChars` method in chunks of 100 characters.

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACCommand cmd = new SACCommand("SELECT * FROM MarketingInformation",
conn);
SADataReader reader = cmd.ExecuteReader();

int idValue;
int productIdValue;
int length = 100;
char[] buf = new char[length];
while (reader.Read())
{
    idValue = reader.GetInt32(0);
    productIdValue = reader.GetInt32(1);
    long blobLength = 0;
    long charsRead;
    while ((charsRead = reader.GetChars(2, blobLength, buf, 0,
length)
        == (long)length)
        {
            blobLength += charsRead;
        }
    blobLength += charsRead;
}
reader.Close();
conn.Close();
```

Time Values

The .NET Framework does not have a `Time` structure. To fetch time values from SAP Sybase IQ, you must use the `GetTimeSpan` method. This method returns the data as a .NET Framework `TimeSpan` object.

C# TimeSpan Example

The following example uses the `GetTimeSpan` method to return the time as `TimeSpan`.

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACCommand cmd = new SACCommand("SELECT 123, CURRENT TIME", conn);
SADataReader reader = cmd.ExecuteReader();
while (reader.Read())
```

```

{
    int ID = reader.GetInt32(0);
    TimeSpan time = reader.GetTimeSpan(1);
}
reader.Close();
conn.Close();

```

Stored Procedures

You can use SQL stored procedures with the SAP Sybase IQ .NET Data Provider.

The `ExecuteReader` method is used to call stored procedures that return result sets, while the `ExecuteNonQuery` method is used to call stored procedures that do not return any result sets. The `ExecuteScalar` method is used to call stored procedures that return only a single value.

You can use `SAPParameter` objects to pass parameters to a stored procedure.

C# Stored Procedure Call with Parameters Example

The following example shows two ways to call a stored procedure and pass it a parameter. The example uses an `SADataReader` to fetch the result set returned by the stored procedure.

```

SAPConnection conn = new SAPConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
bool method1 = true;

SAPCommand cmd = new SAPCommand("", conn);
if (method1)
{
    cmd.CommandText = "ShowProductInfo";
    cmd.CommandType = CommandType.StoredProcedure;
}
else
{
    cmd.CommandText = "call ShowProductInfo(?)";
    cmd.CommandType = CommandType.Text;
}

SAPParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Integer;
param.Direction = ParameterDirection.Input;
param.Value = 301;
cmd.Parameters.Add(param);

SADataReader reader = cmd.ExecuteReader();
reader.Read();
int ID = reader.GetInt32(0);
string name = reader.GetString(1);
string description = reader.GetString(2);
decimal price = reader.GetDecimal(6);
reader.Close();

listBox1.BeginUpdate();
listBox1.Items.Add("Name=" + name +
    " Description=" + description + " Price=" + price);
listBox1.EndUpdate();

```

```
conn.Close();
```

Transaction Processing

With the SAP Sybase IQ .NET Data Provider, you can use the `SATransaction` object to group statements together. Each transaction ends with a `COMMIT` or `ROLLBACK`, which either makes your changes to the database permanent or cancels all the operations in the transaction. Once the transaction is complete, you must create a new `SATransaction` object to make further changes. This behavior is different from ODBC and embedded SQL, where a transaction persists after you execute a `COMMIT` or `ROLLBACK` until the transaction is closed.

If you do not create a transaction, the SAP Sybase IQ .NET Data Provider operates in autocommit mode by default. There is an implicit `COMMIT` after each insert, update, or delete, and once an operation is completed, the change is made to the database. In this case, the changes cannot be rolled back.

Isolation Level Settings for Transactions

The database isolation level is used by default for transactions. You can choose to specify the isolation level for a transaction using the `IsolationLevel` property when you begin the transaction. The isolation level applies to all statements executed within the transaction. The SQL Anywhere .NET Data Provider supports snapshot isolation.

The locks that SAP Sybase IQ uses when you execute a SQL statement depend on the transaction's isolation level.

Distributed Transaction Processing

The .NET 2.0 framework introduced a new namespace `System.Transactions`, which contains classes for writing transactional applications. Client applications can create and participate in distributed transactions with one or multiple participants. Client applications can implicitly create transactions using the `TransactionScope` class. The connection object can detect the existence of an ambient transaction created by the `TransactionScope` and automatically enlist. The client applications can also create a `CommittableTransaction` and call the `EnlistTransaction` method to enlist. This feature is supported by the SAP Sybase IQ .NET Data Provider. Distributed transaction has significant performance overhead. It is recommended that you use database transactions for non-distributed transactions.

C# SATransaction Example

The following example shows how to wrap an `INSERT` into a transaction so that it can be committed or rolled back. A transaction is created with an `SATransaction` object and linked to the execution of a SQL statement using an `SACommand` object. Isolation level 2 (`RepeatableRead`) is specified so that other database users cannot update the row. The lock on the row is released when the transaction is committed or rolled back. If you do not use a transaction, the SAP Sybase IQ .NET Data Provider operates in autocommit mode and you cannot roll back any changes that you make to the database.

```

SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
string stmt = "UPDATE Products SET UnitPrice = 2000.00 " +
    "WHERE Name = 'Tee shirt'";
bool goAhead = false;

SATransaction trans =
conn.BeginTransaction(SAIsolationLevel.RepeatableRead);
SACommand cmd = new SACommand(stmt, conn, trans);
int rowsAffected = cmd.ExecuteNonQuery();
if (goAhead)
    trans.Commit();
else
    trans.Rollback();
conn.Close();

```

Error handling

Your application should be designed to handle any errors that occur.

The SAP Sybase IQ .NET Data Provider creates an SAException object and throws an exception whenever errors occur during execution. Each SAException object consists of a list of SAError objects, and these error objects include the error message and code.

Errors are different from conflicts. Conflicts arise when changes are applied to the database. Your application should include a process to compute correct values or to log conflicts when they arise.

C# error handling example

The following C# code creates a button click handler that opens a connection to the SAP Sybase IQ sample database. If the connection cannot be made, the exception handler displays one or more messages.

```

private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=Sybase IQ
Demo");
    try
    {
        conn.Open();
    }
    catch (SAException ex)
    {
        for (int i = 0; i < ex.Errors.Count; i++)
        {
            MessageBox.Show(ex.Errors[i].Source + " : " +
                ex.Errors[i].Message + " (" +
                ex.Errors[i].NativeError.ToString() + ")",
                "Failed to connect");
        }
    }
}

```

Visual Basic error handling example

The following Visual Basic code creates a button click handler that opens a connection to the SAP Sybase IQ sample database. If the connection cannot be made, the exception handler displays one or more messages.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
    Try
        conn.Open()
    Catch ex As SAException
        For i = 0 To ex.Errors.Count - 1
            MessageBox.Show(ex.Errors(i).Source & " : " & _
                ex.Errors(i).Message & " (" & _
                ex.Errors(i).NativeError.ToString() & ")", _
                "Failed to connect")
        Next i
    End Try
End Sub
```

Entity Framework Support

The SAP Sybase IQ .NET Data Provider supports Entity Framework 4.3, a separate package available from Microsoft. To use Entity Framework 4.3, you must add it to Visual Studio using Microsoft's NuGet Package Manager.

One of the new features of Entity Framework is Code First. It enables a different development workflow: defining data model objects by simply writing C# or VB.NET classes mapping to database objects without ever having to open a designer or define an XML mapping file.

Optionally, additional configuration can be performed by using data annotations or the Fluent API. Models can be used to generate a database schema or map to an existing database.

Here's an example which creates new database objects using the model:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table("EdmCategories", Schema = "DBA")]
    public class Category
    {
        public string CategoryID { get; set; }
        [MaxLength(64)]
        public string Name { get; set; }

        public virtual ICollection<Product> Products { get; set; }
    }
}
```

```

[Table( "EdmProducts", Schema = "DBA" )]
public class Product
{
    public int ProductId { get; set; }
    [MaxLength( 64 )]
    public string Name { get; set; }
    public string CategoryID { get; set; }

    public virtual Category Category { get; set; }
}

[Table( "EdmSuppliers", Schema = "DBA" )]
public class Supplier
{
    [Key]
    public string SupplierCode { get; set; }
    [MaxLength( 64 )]
    public string Name { get; set; }
}

public class Context : DbContext
{
    public Context() : base() { }
    public Context( string connStr ) : base( connStr ) { }

    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }

    protected override void OnModelCreating( DbModelBuilder
modelBuilder )
    {
        modelBuilder.Entity<Supplier>().Property( s =>
s.Name ).IsRequired();
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new
SAConnectionFactory();
            Database.SetInitializer<Context>( new
DropCreateDatabaseAlways<Context>() );

            using ( var db = new Context( "DSN=Sybase IQ Demo" ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}

```

To build and run this example, the following assembly references must be added:

```
EntityFramework
iAnywhere.Data.SQLite.v4.0
System.ComponentModel.DataAnnotations
System.Data.Entity
```

Here is another example that maps to an existing database:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLite;

namespace CodeFirstExample
{
    [Table( "Customers", Schema = "GROUPO" )]
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

        public virtual ICollection<Contact> Contacts { get; set; }
    }

    [Table( "Contacts", Schema = "GROUPO" )]
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }
}
```



```

public class Context : DbContext
{
    public Context() : base() { }
    public Context( string connStr ) : base( connStr ) { }

    public DbSet<Contact> Contacts { get; set; }
    public DbSet<Customer> Customers { get; set; }
}

class Program
{
    static void Main( string[] args )
    {
        Database.DefaultConnectionFactory = new
SAConnectionFactory();
        Database.SetInitializer<Context>( null );

        using ( var db = new Context( "DSN=SAP Sybase IQ 16
Demo" ) )
        {
            foreach ( var customer in db.Customers.ToList() )
            {
                Console.WriteLine( "Customer - " + string.Format(
"{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}",
customer.ID, customer.SurName,
customer.GivenName,
customer.Street, customer.City, customer.State,
customer.Country, customer.PostalCode,
customer.Phone, customer.CompanyName ) );

                foreach ( var contact in customer.Contacts )
                {
                    Console.WriteLine( "    Contact - " +
string.Format(
"{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}, {10}",
contact.ID, contact.SurName,
contact.GivenName,
contact.State,
contact.Title,
contact.Street, contact.City,
contact.State,
contact.Country, contact.PostalCode,
contact.Phone, contact.Fax ) );
                }
            }
        }
    }
}

```

There are some implementation detail differences between the Microsoft .NET Framework Data Provider for SQL Server (SqlClient) and the SAP Sybase IQ .NET Data Provider of which you should be aware.

1. A new class `SACConnectionFactory` (implements `IDbConnectionFactory`) is included. You set the `Database.DefaultConnectionFactory` to an instance of `SACConnectionFactory` before creating any data model as shown below:

```
Database.DefaultConnectionFactory = new SACConnectionFactory();
```

2. The major principle of Entity Framework Code First is coding by conventions. The Entity Framework infers the data model by coding conventions. Entity Framework also does lots of things implicitly. Sometimes the developer might not realize all these Entity Framework conventions. But some code conventions do not make sense for SAP Sybase IQ. There is a big difference between SQL Server and SAP Sybase IQ. Every SQL Server instance maintains multiple databases, but every SAP Sybase IQ database is a single file.

- If the user creates a user-defined `DbContext` using the parameterless constructor, `SqlClient` will connect to SQL Server Express on the local computer using integrated security. The SAP Sybase IQ provider connects to the default server using integrated login if the user has already created a login mapping.
- `SqlClient` drops the existing database and creates a new database when the Entity Framework calls `DbDeleteDatabase` or `DbCreateDatabase` (SQL Server Express Edition only). The SAP Sybase IQ provider never drops or creates the database. It creates or drops the database objects (tables, relations, constraints for example). The user must create the database first.
- The `IDbConnectionFactory.CreateConnection` method treats the string parameter "nameOrConnectionString" as database name (initial catalog for SQL Server) or a connection string. If the user does not provide the connection string for `DbContext`, `SqlClient` will automatically connect to the SQL Express server on the local computer using the namespace of user-defined `DbContext` class as the initial catalog. For SAP Sybase IQ, that parameter can only contain a connection string. A database name will be ignored and integrated login will be used instead.

3. The SQL Server `SqlClient` API maps a column with data annotation attribute `TimeStamp` to SQL Server data type `timestamp/rowversion`. There are some misconceptions about SQL Server `timestamp/rowversion` among developers. The SQL Server `timestamp/rowversion` data type is different from SAP Sybase IQ and most other database vendors:

- The SQL Server `timestamp/rowversion` is binary(8). It does not support a combined date and time value. SAP Sybase IQ supports a data type called `timestamp` that is equivalent to the SQL Server `datetime` data type.
- SQL Server `timestamp/rowversion` values are guaranteed to be unique. SAP Sybase IQ `timestamp` values are not unique.
- A SQL Server `timestamp/rowversion` value changes every time the row is updated.

The `TimeStamp` data annotation attribute is not supported by SAP Sybase IQ provider.

4. By default, Entity Framework 4.1 always sets the schema or owner name to `dbo` which is the default schema of SQL Server. However, `dbo` is not appropriate for SAP Sybase IQ. For SAP Sybase IQ, you must specify the schema name (`GROUP0` for example) with the table name either by using data annotations or the Fluent API. Here's an example:

```
namespace CodeFirstTest
{
    public class Customer
```

```

{
    [Key()]
    public int ID { get; set; }
    public string SurName { get; set; }
    public string GivenName { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Country { get; set; }
    public string PostalCode { get; set; }
    public string Phone { get; set; }
    public string CompanyName { get; set; }

    public virtual ICollection<Contact> Contacts { get; set; }
}

public class Contact
{
    [Key()]
    public int ID { get; set; }
    public string SurName { get; set; }
    public string GivenName { get; set; }
    public string Title { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Country { get; set; }
    public string PostalCode { get; set; }
    public string Phone { get; set; }
    public string Fax { get; set; }

    [ForeignKey( "Customer" )]
    public int CustomerID { get; set; }
    public virtual Customer Customer { get; set; }
}

[Table( "Departments", Schema = "GROUPO" )]
public class Department
{
    [Key()]
    public int DepartmentID { get; set; }
    public string DepartmentName { get; set; }
    public int DepartmentHeadID { get; set; }
}

public class Context : DbContext
{
    public Context() : base() { }
    public Context( string connStr ) : base( connStr ) { }

    public DbSet<Contact> Contacts { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Department> Departments { get; set; }

    protected override void OnModelCreating( DbModelBuilder
modelBuilder )

```

```
{
    modelBuilder.Entity<Contact>().ToTable( "Contacts",
"GROUPO" );
    modelBuilder.Entity<Customer>().ToTable( "Customers",
"GROUPO" );
}
}
```

SAP Sybase IQ .NET Data Provider Deployment

The following sections describe how to deploy the SAP Sybase IQ .NET Data Provider.

SAP Sybase IQ .NET Data Provider System Requirements

To use the SAP Sybase IQ .NET Data Provider, you must have the following installed on your computer or handheld device:

- The .NET Framework and/or .NET Compact Framework version 2.0 or later.
- Visual Studio 2005 or later, or a .NET language compiler, such as C# (required only for development).

SAP Sybase IQ .NET Data Provider Required Files

The SAP Sybase IQ .NET Data Provider code resides in a DLL for each platform.

Windows Required Files

For Windows, one of the following DLLs is required:

- %IQDIR16%\V2\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll
- %IQDIR16%\V2\Assembly
 \V3.5\iAnywhere.Data.SQLAnywhere.v3.5.dll
- %IQDIR16%\V2\Assembly
 \V4\iAnywhere.Data.SQLAnywhere.v4.0.dll

The choice of DLL depends on the version of .NET that you are targeting.

The Windows version of the provider also requires the following DLLs.

- **policy.16.0.iAnywhere.Data.SQLAnywhere.dll** – The policy file can be used to override the provider version that the application was built with. The policy file is updated by Sybase whenever an update to the provider is released. There are also policy files for the version 3.5 provider (`policy.16.0.iAnywhere.Data.SQLAnywhere.v3.5.dll`) and the version 4.0 provider (`policy.16.0.iAnywhere.Data.SQLAnywhere.v4.0.dll`).
- **db1gen16.dll** – This language DLL contains English (en) messages issued by the provider. It is available in many other languages including Chinese (zh), French (fr), German (de), and Japanese (jp).

- **dbcon16.dll** – The **Connect to SQL Anywhere** window support code is contained in this DLL.

Visual Studio deploys the .NET Data Provider DLL

(`iAnywhere.Data.SQLAnywhere.dll` or

`iAnywhere.Data.SQLAnywhere.v3.5.dll`) to your device along with your

program. If you are not using Visual Studio, you must copy the Data Provider DLL to the device along with your application. It can go in the same directory as your application, or in the `\Windows` directory.

The SAP Sybase IQ .NET Data Provider dbdata DLL

When the SAP Sybase IQ .NET Data Provider is first loaded by a .NET application (usually when making a database connection using `SACConnection`), it unpacks a DLL that contains the provider's unmanaged code. The file `dbdata16.dll` is placed by the provider in a subdirectory of the directory identified using the following strategy.

1. The first directory it attempts to use for unloading is the one returned by the first of the following:
 - The path identified by the `TMP` environment variable.
 - The path identified by the `TEMP` environment variable.
 - The path identified by the `USERPROFILE` environment variable.
 - The Windows directory.
2. If the identified directory is inaccessible, then the provider will attempt to use the current working directory.
3. If the current working directory is inaccessible, then the provider will attempt to use the directory from where the application itself was loaded.

The subdirectory name will take the form of a GUID with a suffix including the version number, bitness of the DLL, and an index number used to guarantee uniqueness. The following is an example of a possible subdirectory name.

```
{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_1601.x64_1
```

SAP Sybase IQ .NET Data Provider DLL Registration

The Windows version of the SAP Sybase IQ .NET Data Provider DLL (`%IQDIR%\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll`) is registered in the Global Assembly Cache when you install the SAP Sybase IQ software. On Windows Mobile, you do not need to register the DLL.

If you are deploying the SAP Sybase IQ .NET Data Provider, you can register it using the `gacutil` utility that is included with the Microsoft SDK.

To register the SAP Sybase IQ .NET Data Provider as a `DbProviderFactory` instance when deploying the provider, you must add an entry to the `.NETmachine.config` file. An entry similar to the following must be placed in the `<DbProviderFactories>` section.

```
<add invariant="iAnywhere.Data.SQLAnywhere"
name="SAP Sybase IQ 16 Data Provider"
```

.NET Application Programming

```
description=".Net Framework Data Provider for SAP Sybase IQ 16"  
type="iAnywhere.Data.SQLAnywhere.SAFactory,  
iAnywhere.Data.SQLAnywhere.v3.5,  
Version=16.0.0.36003, Culture=neutral,  
PublicKeyToken=f222fc4333e0d400"/>
```

The version number must match the version of the provider that you are installing. The configuration file is located in `\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`. For 64-bit Windows systems, there is a second configuration file under the `Framework64` tree that must also be modified.

.NET Tracing Support

The SAP Sybase IQ .NET Data Provider supports tracing using the .NET tracing feature.

By default, tracing is disabled. To enable tracing, specify the trace source in your application's configuration file.

The following is an example of a configuration file:

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
<system.diagnostics>  
<sources>  
  <source name="iAnywhere.Data.SQLAnywhere"  
    switchName="SASourceSwitch"  
    switchType="System.Diagnostics.SourceSwitch">  
    <listeners>  
      <add name="ConsoleListener"  
        type="System.Diagnostics.ConsoleTraceListener"/>  
      <add name="EventListener"  
        type="System.Diagnostics.EventLogTraceListener"  
        initializeData="MyEventLog"/>  
      <add name="TraceLogListener"  
        type="System.Diagnostics.TextWriterTraceListener"  
        initializeData="myTrace.log"  
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>  
      <remove name="Default"/>  
    </listeners>  
  </source>  
</sources>  
<switches>  
  <add name="SASourceSwitch" value="All"/>  
  <add name="SATraceAllSwitch" value="1" />  
  <add name="SATraceExceptionSwitch" value="1" />  
  <add name="SATraceFunctionSwitch" value="1" />  
  <add name="SATracePoolingSwitch" value="1" />  
  <add name="SATracePropertySwitch" value="1" />  
</switches>  
</system.diagnostics>  
</configuration>
```

There are four types of trace listeners referenced in the configuration file shown above.

- **ConsoleTraceListener** – Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the **Output** window.
- **DefaultTraceListener** – This listener is automatically added to the `Debug.Listeners` and `Trace.Listeners` collections using the name "Default". Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the **Output** window. To avoid duplication of output produced by the `ConsoleTraceListener`, this listener is removed.
- **EventLogTraceListener** – Tracing or debugging output is directed to an `EventLog` identified in the `initializeData` option. In the example, the event log is named `MyEventLog`. Writing to the system event log requires administrator privileges and is not a recommended method for debugging applications.
- **TextWriterTraceListener** – Tracing or debugging output is directed to a `TextWriter` which writes the stream to the file identified in the `initializeData` option.

To disable tracing to any of the trace listeners described above, remove the corresponding `add` entry under `<listeners>`.

The trace configuration information is placed in the application's project folder in the `App.config` file. If the file does not exist, it can be created and added to the project using Visual Studio by choosing **Add » New Item** and selecting **Application Configuration File**.

The `traceOutputOptions` can be specified for any listener and include the following:

- **Callstack** – Write the call stack, which is represented by the return value of the `Environment.StackTrace` property.
- **DateTime** – Write the date and time.
- **LogicalOperationStack** – Write the logical operation stack, which is represented by the return value of the `CorrelationManager.LogicalOperationStack` property.
- **None** – Do not write any elements.
- **ProcessId** – Write the process identity, which is represented by the return value of the `Process.Id` property.
- **ThreadId** – Write the thread identity, which is represented by the return value of the `Thread.ManagedThreadId` property for the current thread.
- **Timestamp** – Write the timestamp, which is represented by the return value of the `System.Diagnostics.Stopwatch.GetTimeStamp` method.

The example configuration file, shown earlier, specifies trace output options for the `TextWriterTraceListener` only.

You can limit what is traced by setting specific trace options. By default the numeric-valued trace option settings are all 0. The trace options that can be set include the following:

- **SASourceSwitch** – `SASourceSwitch` can take any of the following values. If it is `Off` then there is no tracing.

Off – Does not allow any events through.

Critical – Allows only Critical events through.

Error – Allows Critical and Error events through.

Warning – Allows Critical, Error, and Warning events through.

Information – Allows Critical, Error, Warning, and Information events through.

Verbose – Allows Critical, Error, Warning, Information, and Verbose events through.

ActivityTracing – Allows the Stop, Start, Suspend, Transfer, and Resume events through.

All – Allows all events through.

Here is an example setting.

```
<add name="SASourceSwitch" value="Error"/>
```

- **SATraceAllSwitch** – All the trace options are enabled. You do not need to set any other options since they are all selected. You cannot disable individual options if you choose this option. For example, the following will not disable exception tracing.

```
<add name="SATraceAllSwitch" value="1" />  
<add name="SATraceExceptionSwitch" value="0" />
```

- **SATraceExceptionSwitch** – All exceptions are logged. Trace messages have the following form.

```
<Type|ERR> message='message_text' [ nativeError=error_number]
```

The nativeError=error_number text will only be displayed if there is an SAException object.

- **SATraceFunctionSwitch** – All function scope entry/exits are logged. Trace messages have any of the following forms.

```
enter_nnn <sa.class_name.method_name|API> [object_id#]  
[parameter_names]  
leave_nnn
```

The nnn is an integer representing the scope nesting level 1, 2, 3, ... The optional parameter_names is a list of parameter names separated by spaces.

- **SATracePoolingSwitch** – All connection pooling is logged. Trace messages have any of the following forms.

```
<sa.ConnectionPool.AllocateConnection|CPOOL>  
connectionString='connection_text'  
<sa.ConnectionPool.RemoveConnection|CPOOL>  
connectionString='connection_text'  
<sa.ConnectionPool.ReturnConnection|CPOOL>  
connectionString='connection_text'  
<sa.ConnectionPool.ReuseConnection|CPOOL>  
connectionString='connection_text'
```

- **SATracePropertySwitch** – All property setting and retrieval is logged. Trace messages have any of the following forms.


```
<sa.class_name.get_property_name|API> object_id#
<sa.class_name.set_property_name|API> object_id#
```

For more information, see "Tracing Data Access" at <http://msdn.microsoft.com/en-us/library/ms971550.aspx>.

Configuring a Windows Application for Tracing

Enabling tracing on the TableViewer sample application involves creating a configuration file that references the ConsoleTraceListener and TextWriterTraceListener listeners, removes the default listener, and enables all switches that would otherwise be set to 0.

Prerequisites

You must have Visual Studio installed.

Task

1. Open the TableViewer sample in Visual Studio.

Start Visual Studio and open the %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\TableViewer\TableViewer.sln.

2. Create an application file named App.config and copy the following configuration setup:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
  <source name="iAnywhere.Data.SQLAnywhere"
    switchName="SASourceSwitch"
    switchType="System.Diagnostics.SourceSwitch">
    <listeners>
      <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
      <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
      <remove name="Default"/>
    </listeners>
    </source>
  </sources>
<switches>
  <add name="SASourceSwitch" value="All"/>
  <add name="SATraceAllSwitch" value="1" />
  <add name="SATraceExceptionSwitch" value="1" />
  <add name="SATraceFunctionSwitch" value="1" />
  <add name="SATracePoolingSwitch" value="1" />
  <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

3. Rebuild the application.
4. Click **Debug** » **Start Debugging**.

When the application finishes execution, the trace output is recorded in the `bin\Debug\myTrace.log` file.

Next

View the trace log in the **Output** window of Visual Studio.

.NET Data Provider Tutorials

The Simple and Table Viewer sample projects are included with the .NET Data Provider.

The sample projects can be used with Visual Studio 2005 or later versions. The sample projects were developed with Visual Studio 2005. If you use a later version, you may have to run the Visual Studio **Upgrade Wizard**. This section also includes a tutorial that takes you through the steps of building the Simple Viewer .NET database application using Visual Studio.

Tutorial: Using the Simple Code Sample

The Simple project uses the .NET Data Provider to obtain a result set from the database server.

Prerequisites

You must have Visual Studio and the .NET Framework installed on your computer.

You must have the `SELECT ANY TABLE` system privilege.

Task

The Simple project is included with the SAP Sybase IQ samples. It demonstrates a simple listbox that is filled with the names from the Employees table.

1. Start Visual Studio.
2. Click **File** » **Open** » **Project**.
3. Browse to `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleWin32` and open the `Simple.sln` project.
4. When you use the SAP Sybase IQ .NET Data Provider in a project, you must add a reference to the Data Provider. This has already been done in the Simple code sample. To view the reference to the Data Provider (`iAnywhere.Data.SQLAnywhere`), open the **References** folder in the **Solution Explorer** window.

5. You must also add a `using` directive to your source code to reference the Data Provider classes. This has already been done in the Simple code sample. To view the `using` directive:

- Open the source code for the project. In the **Solution Explorer** window, right-click `Form1.cs` and click **View Code**.

In the `using` directives in the top section, you should see the following line:

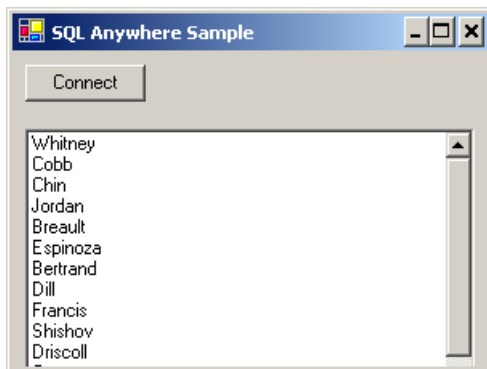
```
using iAnywhere.Data.SQLAnywhere;
```

This line is required for C# projects. If you are using Visual Basic .NET, you need to add an `Imports` line to your source code.

6. Click **Debug » Start Without Debugging** or press `Ctrl+F5` to run the Simple sample.

7. In the **SQL Anywhere Sample** window, click **Connect**.

The application connects to the SAP Sybase IQ sample database and puts the surname of each employee in the window, as follows:



8. Close the **SQL Anywhere Sample** window to shut down the application and disconnect from the sample database. This also shuts down the database server.

You have built and executed a simple .NET application that uses the SAP Sybase IQ .NET Data Provider to obtain a result set from an SAP Sybase IQ database server.

Tutorial: Using the Table Viewer Code Sample

The `TableViewer` project uses the .NET Data Provider to connect to a database, execute SQL statements, and display the results using a `DataGrid` object.

Prerequisites

You must have Visual Studio and the .NET Framework installed on your computer.

You must have the `SELECT ANY TABLE` system privilege.

Task

The TableViewer project is included with the SAP Sybase IQ samples. The Table Viewer project is more complex than the Simple project. You can use it to connect to a database, select a table, and execute SQL statements on the database.

1. Start Visual Studio.
2. Click **File » Open » Project**.
3. Browse to `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\TableViewer` and open the `TableViewer.sln` project.
4. To use the SAP Sybase IQ .NET Data Provider in a project, you must add a reference to the Data Provider DLL. This has already been done in the Table Viewer code sample. To view the reference to the Data Provider (`iAnywhere.Data.SQLAnywhere`), open the **References** folder in the **Solution Explorer** window.
5. You must also add a `using` directive to your source code to reference the Data Provider classes. This has already been done in the Table Viewer code sample. To view the `using` directive:

- Open the source code for the project. In the **Solution Explorer** window, right-click `TableViewer.cs` and click **View Code**.
- In the `using` directives in the top section, you should see the following line:

```
using iAnywhere.Data.SQLAnywhere;
```

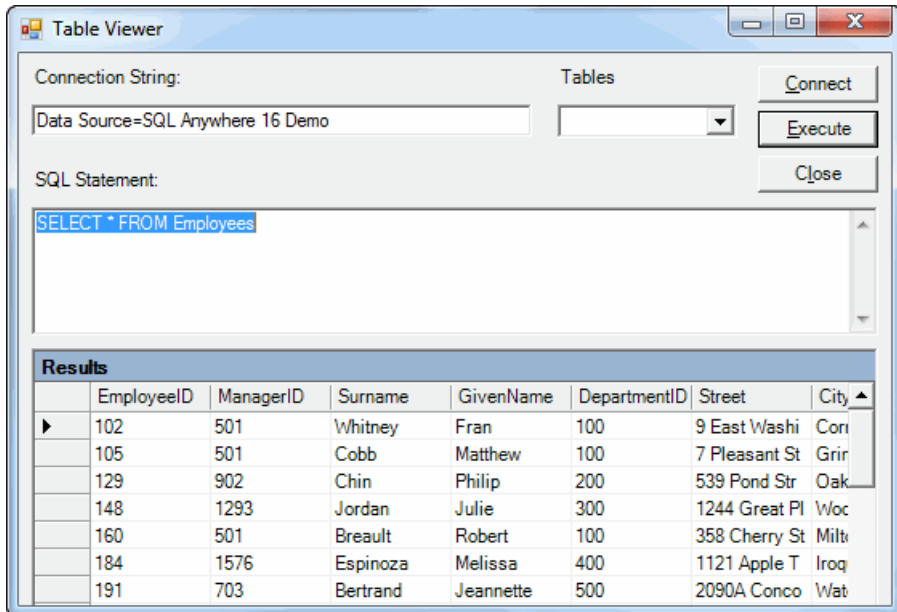
This line is required for C# projects. If you are using Visual Basic, you need to add an `Imports` line to your source code.

6. Click **Debug » Start Without Debugging** or press `Ctrl+F5` to run the Table Viewer sample.

The application connects to the SAP Sybase IQ sample database.

7. In the **Table Viewer** window, click **Connect**.
8. In the **Table Viewer** window, click **Execute**.

The application retrieves the data from the Employees table in the sample database and puts the query results in the **Results** datagrid, as follows:



You can also execute other SQL statements from this application: type a SQL statement in the **SQL Statement** pane, and then click **Execute**.

9. Close the **Table Viewer** window to shut down the application and disconnect from the sample database. This also shuts down the database server.

You have built and executed a .NET application that uses the .NET Data Provider to connect to a database, execute SQL statements, and display the results using a DataGrid object.

Tutorial: Developing a Simple .NET Database Application with Visual Studio

This section contains a tutorial that takes you through the steps of building the Simple Viewer .NET database application using Visual Studio.

Prerequisites

You must have the SELECT ANY TABLE system privilege.

Lesson 1: Creating a Table Viewer

In this lesson, you use Microsoft Visual Studio, the Server Explorer, and the SAP Sybase IQ .NET Data Provider to create an application that accesses one of the tables in the SAP Sybase IQ sample database, allowing you to examine rows and perform updates.

Prerequisites

You must have Visual Studio and the .NET Framework installed on your computer.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Developing a simple .NET database application with Visual Studio.

Task

This tutorial is based on Visual Studio and the .NET Framework. The complete application can be found in the ADO.NET project %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln.

1. Start Visual Studio.
2. Click **File » New » Project**.

The **New Project** window appears.

- a. In the left pane of the **New Project** window, click either **Visual Basic** or **Visual C#** for the programming language.
 - b. From the **Windows** subcategory, click **Windows Application** (VS 2005) or **Windows Forms Application** (VS 2008/2010).
 - c. In the project **Name** field, type MySimpleViewer.
 - d. Click **OK** to create the new project.
3. Click **View » Server Explorer**.
 4. In the **Server Explorer** window, right-click **Data Connections** and click **Add Connection**.
 5. In the **Add Connection** window:

- a. If you have never used **Add Connection** for other projects, then you see a list of data sources. Click **SQL Anywhere** from the list of data sources presented.
If you have used **Add Connection** before, then click **Change** to change the data source to **SQL Anywhere**.
- b. Under **Data Source**, click **ODBC Data Source Name** and type Sybase IQ Demo.

Note: When using the Visual Studio Add Connection wizard on 64-bit Windows, only the 64-bit System Data Source Names (DSN) are included with the User Data Source Names. Any 32-bit System Data Source Names are not displayed. In Visual Studio's 32-bit design environment, the Test Connection button will attempt to establish a connection using the 32-bit equivalent of the 64-bit System DSN. If the 32-bit System DSN does not exist, the test will fail.

- c. Click **Test Connection** to verify that you can connect to the sample database.
- d. Click **OK**.

A new connection named Sybase IQ.demo appears in the **Server Explorer** window.

6. Expand the Sybase IQ.demo connection in the **Server Explorer** window until you see the table names.

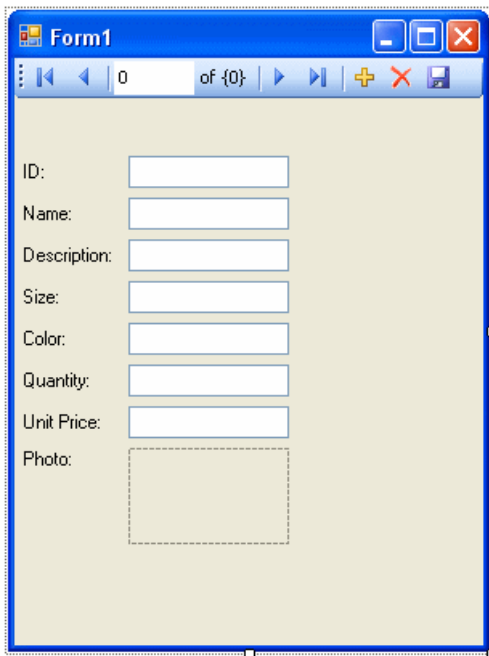
(Visual Studio 2005 only) Try the following:

- a. Right-click the Products table and click **Show Table Data**.
This shows the rows and columns of the Products table in a window.
 - b. Close the table data window.
7. Click **Data » Add New Data Source**.
8. In the **Data Source Configuration Wizard**, do the following:
- a. On the **Data Source Type** page, click **Database**, then click **Next**.
 - b. (Visual Studio 2010 only) On the **Database Model** page, click **Dataset**, then click **Next**.
 - c. On the **Data Connection** page, click `Sybase IQ.demo`, then click **Next**.
 - d. On the **Save The Connection String** page, make sure that **Yes, Save The Connection As** is chosen and click **Next**.
 - e. On the **Choose Your Database Objects** page, click **Tables**, then click **Finish**.
9. Click **Data » Show Data Sources**.

The **Data Sources** window appears.

Expand the Products table in the **Data Sources** window.

- a. Click Products, then click **Details** from the dropdown list.
- b. Click Photo, then click **Picture Box** from the dropdown list.
- c. Click Products and drag it to your form (Form1).



The screenshot shows a Windows Form titled "Form1" with a standard Windows XP-style title bar. Below the title bar is a toolbar with navigation and editing icons. The main area of the form contains a data-bound form with the following fields:

- ID:
- Name:
- Description:
- Size:
- Color:
- Quantity:
- Unit Price:
- Photo:

The Photo field is represented by a dashed rectangular box, indicating it is a picture box control.

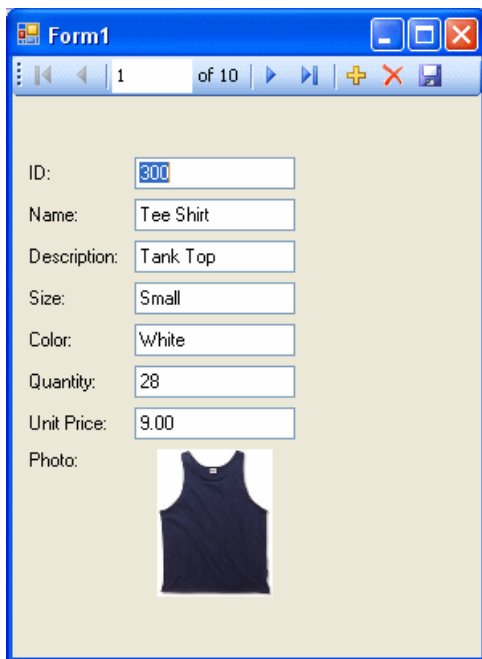
A dataset control and several labeled text fields appear on the form.

10. On the form, click the picture box next to Photo.
 - a. Change the shape of the box to a square.
 - b. Click the right-arrow in the upper-right corner of the picture box.
The **Picture Box Tasks** window opens.
 - c. From the **Size Mode** dropdown list, click **Zoom**.
 - d. To close the **Picture Box Tasks** window, click anywhere outside the window.

11. Build and run the project.

- a. Click **Build » Build Solution**.
- b. Click **Debug » Start Debugging**.

The application connects to the SAP Sybase IQ sample database and displays the first row of the Products table in the text boxes and picture box.



- c. You can use the buttons on the control to scroll through the rows of the result set.
- d. You can go directly to a row in the result set by entering the row number in the scroll control.
- e. You can update values in the result set using the text boxes and save them by clicking the **Save Data** button.

12. Shut down the application and then save your project.

You have now created a simple, yet powerful, .NET application using Visual Studio, the Server Explorer, and the SAP Sybase IQ .NET Data Provider.

Next

In the next lesson, you add a datagrid control to the form developed in this lesson.

Lesson 2: Adding a Synchronizing Data Control

In this lesson, you add a datagrid control to the form developed in the previous lesson. This control updates automatically as you navigate through the result set.

Prerequisites

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using Java in the databas Tutorial: Developing a Simple .NET Database Application with Visual Studio.

Task

The complete application can be found in the ADO.NET project %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln.

1. Start Visual Studio and load your MySimpleViewer project.
2. Right-click DataSet1 in the **Data Sources** window and click **Edit DataSet With Designer**.
3. Right-click an empty area in the **DataSet Designer** window and click **Add » TableAdapter**.
4. In the **TableAdapter Configuration Wizard**:
 - a. On the **Choose Your Data Connection** page, click **Next**.
 - b. On the **Choose A Command Type** page, click **Use SQL Statements**, then click **Next**.
 - c. On the **Enter A SQL Statement** page, click **Query Builder**.
 - d. On the **Add Table** window, click the **Views** tab, then click **ViewSalesOrders**, and then click **Add**.
 - e. Click **Close** to close the **Add Table** window.
5. Expand the **Query Builder** window so that all sections of the window are visible.
 - a. Expand the **ViewSalesOrders** window so that all the checkboxes are visible.
 - b. Click **Region**.
 - c. Click **Quantity**.
 - d. Click **ProductID**.
 - e. In the grid below the **ViewSalesOrders** window, clear the checkbox under **Output** for the ProductID column.
 - f. For the ProductID column, type a question mark (?) in the **Filter** cell. This generates a WHERE clause for ProductID.

A SQL query has been built that looks like the following:

.NET Application Programming

```
SELECT Region, Quantity
FROM GROUPO.ViewSalesOrders
WHERE (ProductID = :Param1)
```

6. Modify the SQL query as follows:

- a. Change Quantity to SUM(Quantity) AS TotalSales.
- b. Add GROUP BY Region to the end of the query following the WHERE clause.

The modified SQL query now looks like this:

```
SELECT Region, SUM(Quantity) as TotalSales
FROM GROUPO.ViewSalesOrders
WHERE (ProductID = :Param1)
GROUP BY Region
```

7. Click **OK**.

8. Click **Finish**.

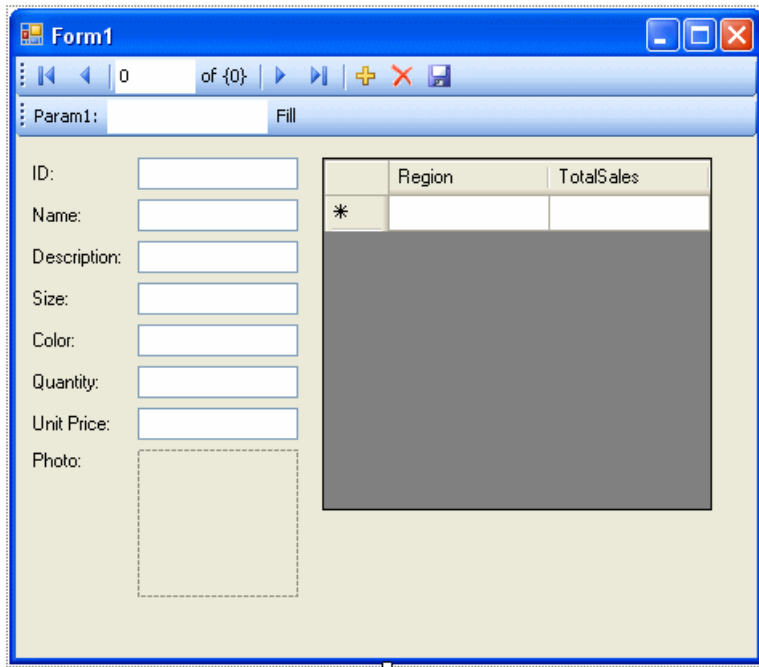
A new **TableAdapter** called **ViewSalesOrders** has been added to the **DataSet Designer** window.

9. Click the form design tab (Form1).

- Stretch the form to the right to make room for a new control.

10. Expand ViewSalesOrders in the **Data Sources** window.

- a. Click ViewSalesOrders and click **DataGridView** from the dropdown list.
- b. Click ViewSalesOrders and drag it to your form (Form1).

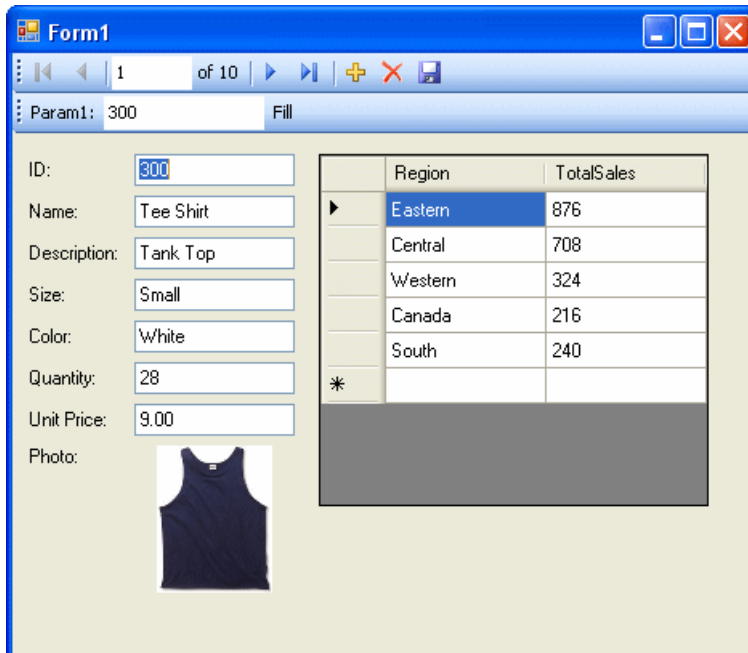


A datagrid view control appears on the form.

11. Build and run the project.

- Click **Build » Build Solution**.
- Click **Debug » Start Debugging**.
- In the **Param1** or **ProductID** (VS 2010) text box, enter a product ID number such as 300 and click **Fill**.

The datagrid view displays a summary of sales by region for the product ID entered.



You can also use the other control on the form to move through the rows of the result set.

It would be ideal, however, if both controls could stay synchronized with each other. The next few steps show how to do this.

12. Shut down the application and then save your project.

13. Delete the Fill strip on the form since you do not need it.

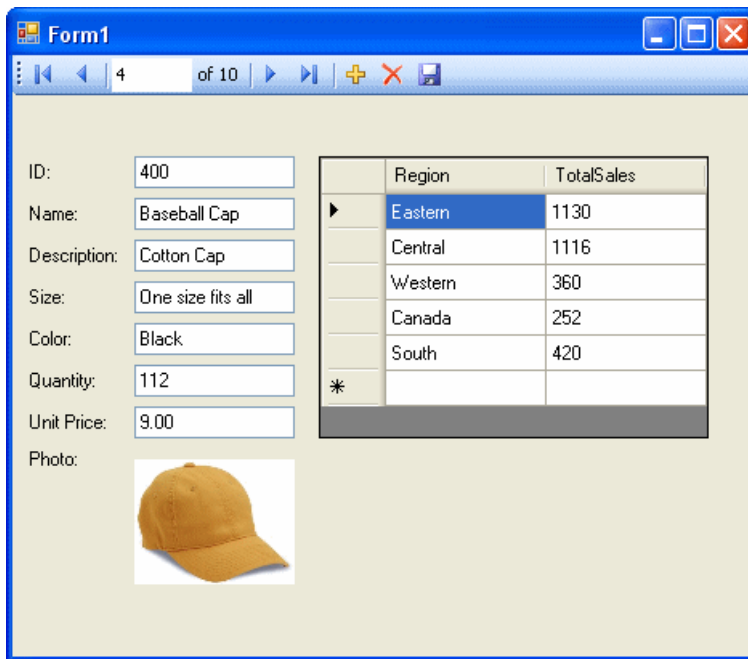
- On the design form (Form1), right-click the Fill strip to the right of the word **Fill**, then click **Delete**.

The Fill strip is removed from the form.

14. Synchronize the two controls as follows.

- a. On the design form (Form1), right-click the ID text box, then click **Properties**.
- b. Click the **Events** button (it appears as a lightning bolt).
- c. Scroll down until you find the **TextChanged** event.

- d. Click **TextChanged**, then click **fillToolStripButton_Click** from the dropdown list. If you are using Visual Basic, the event is called **FillToolStripButton_Click**.
 - e. Double-click **fillToolStripButton_Click** and the form's code window opens on the `fillToolStripButton_Click` event handler.
 - f. Find the reference to `param1ToolStripTextBox` or `productIDToolStripTextBox` (VS 2010) and change this to `IDTextBox`. If you are using Visual Basic, the text box is called `IDTextBox`.
 - g. Rebuild and run the project.
15. The application form now appears with a single navigation control.
- The datagrid view displays an updated summary of sales by region corresponding to the current product as you move through the result set.



16. Shut down the application and then save your project.

You have now added a control that updates automatically as you navigate through the result set.

In this tutorial, you saw how the powerful combination of Microsoft Visual Studio, the Server Explorer, and the SAP Sybase IQ .NET Data Provider can be used to create database applications.

.NET API Reference

The namespace is `iAnywhere.Data.SQLAnywhere`.

SInfoMessageEventHandler(object, SInfoMessageEventArgs) delegate

Represents the method that handles the `SACConnection.InfoMessage` event of an `SACConnection` object.

Visual Basic syntax

```
Public Delegate Sub SInfoMessageEventHandler (ByVal obj As  
Object, ByVal args As SInfoMessageEventArgs )
```

C# syntax

```
public delegate void SInfoMessageEventHandler (object obj,  
SInfoMessageEventArgs args);
```

SARowsCopiedEventHandler(object, SARowsCopiedEventArgs) delegate

Represents the method that handles the `SABulkCopy.SARowsCopied` event of an `SABulkCopy`.

Visual Basic syntax

```
Public Delegate Sub SARowsCopiedEventHandler (ByVal sender As  
Object, ByVal rowsCopiedEventArgs As SARowsCopiedEventArgs )
```

C# syntax

```
public delegate void SARowsCopiedEventHandler (object sender,  
SARowsCopiedEventArgs rowsCopiedEventArgs);
```

Usage

The `SARowsCopiedEventHandler` delegate is not available in the .NET Compact Framework 2.0.

SARowUpdatedEventHandler(object, SARowUpdatedEventArgs) delegate

Represents the method that handles the RowUpdated event of an SADataAdapter.

Visual Basic syntax

```
Public Delegate Sub SARowUpdatedEventHandler (ByVal sender As  
Object, ByVal e As SARowUpdatedEventArgs )
```

C# syntax

```
public delegate void SARowUpdatedEventHandler (object sender,  
SARowUpdatedEventArgs e);
```

SARowUpdatingEventHandler(object, SARowUpdatingEventArgs) delegate

Represents the method that handles the RowUpdating event of an SADataAdapter.

Visual Basic syntax

```
Public Delegate Sub SARowUpdatingEventHandler (ByVal sender As  
Object, ByVal e As SARowUpdatingEventArgs )
```

C# syntax

```
public delegate void SARowUpdatingEventHandler (object sender,  
SARowUpdatingEventArgs e);
```

SABulkCopyOptions() enumeration

A bitwise flag that specifies one or more options to use with an instance of SABulkCopy.

Enum Constant Summary

- **Default** – Specifying only this value causes the default behavior to be used.
- **DoNotFireTriggers** – When specified, triggers are not fired.
- **KeepIdentity** – When specified, the source values to be copied into an identity column are preserved.
- **TableLock** – When specified the table is locked using the command LOCK TABLE table_name WITH HOLD IN SHARE MODE.
- **UseInternalTransaction** – When specified, each batch of the bulk-copy operation is executed within a transaction.

SALsolationLevel() enumeration

Specifies SQL Anywhere isolation levels.

Enum Constant Summary

- **Chaos** – This isolation level is unsupported.
- **ReadCommitted** – Sets the behavior to be equivalent to isolation level 1.
- **ReadUncommitted** – Sets the behavior to be equivalent to isolation level 0.
- **RepeatableRead** – Sets the behavior to be equivalent to isolation level 2.
- **Serializable** – Sets the behavior to be equivalent to isolation level 3.
- **Snapshot** – Uses a snapshot of committed data from the time when the first row is read, inserted, updated, or deleted by the transaction.
- **Unspecified** – This isolation level is unsupported.
- **ReadOnlySnapshot** – For read-only statements, use a snapshot of committed data from the time when the first row is read from the database.
- **StatementSnapshot** – Use a snapshot of committed data from the time when the first row is read by the statement.

SABulkCopy class

Efficiently bulk load a SQL Anywhere table with data from another source.

Visual Basic syntax

```
Public NotInheritable Class SABulkCopy Implements
System.IDisposable
```

C# syntax

```
public sealed class SABulkCopy: System.IDisposable
```

Remarks

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

Implements: System.IDisposable

Custom Attribute: sealed

Close() method

Closes the SABulkCopy instance.

Visual Basic syntax

```
Public Sub Close ()
```

C# syntax

```
public void Close ()
```

BatchSize property

Gets or sets the number of rows in each batch.

Visual Basic syntax

```
Public Property BatchSize As Integer
```

C# syntax

```
public int BatchSize {get;set;}
```

Remarks

At the end of each batch, the rows in the batch are sent to the server.

The number of rows in each batch. The default is 0.

Setting this property to zero causes all the rows to be sent in one batch.

Setting this property to a value less than zero is an error.

If this value is changed while a batch is in progress, the current batch completes and any further batches use the new value.

BulkCopyTimeout property

Gets or sets the number of seconds for the operation to complete before it times out.

Visual Basic syntax

```
Public Property BulkCopyTimeout As Integer
```

C# syntax

```
public int BulkCopyTimeout {get;set;}
```

Remarks

The default value is 30 seconds.

A value of zero indicates no limit. This should be avoided because it may cause an indefinite wait.

If the operation times out, then all rows in the current transaction are rolled back and an `SAException` is raised.

Setting this property to a value less than zero is an error.

ColumnMappings property

Returns a collection of SABulkCopyColumnMapping items.

Visual Basic syntax

```
Public ReadOnly Property ColumnMappings As  
SABulkCopyColumnMappingCollection
```

C# syntax

```
public SABulkCopyColumnMappingCollection ColumnMappings {get;}
```

Remarks

Column mappings define the relationships between columns in the data source and columns in the destination.

By default, it is an empty collection.

The property cannot be modified while WriteToServer is executing.

If ColumnMappings is empty when WriteToServer is executed, then the first column in the source is mapped to the first column in the destination, the second to the second, and so on. This takes place as long as the column types are convertible, there are at least as many destination columns as source columns, and any extra destination columns are nullable.

DestinationTableName property

Gets or sets the name of the destination table on the server.

Visual Basic syntax

```
Public Property DestinationTableName As String
```

C# syntax

```
public string DestinationTableName {get;set;}
```

Remarks

The default value is a null reference. In Visual Basic it is Nothing.

If the value is changed while WriteToServer is executing, the change has no effect.

If the value has not been set before a call to WriteToServer, an InvalidOperationException is raised.

It is an error to set the value to NULL or the empty string.

NotifyAfter property

Gets or sets the number of rows to be processed before generating a notification event.

Visual Basic syntax

```
Public Property NotifyAfter As Integer
```

C# syntax

```
public int NotifyAfter {get;set;}
```

Remarks

Zero is returned if the property has not been set.

Changes made to NotifyAfter, while executing WriteToServer, do not take effect until after the next notification.

Setting this property to a value less than zero is an error.

The values of NotifyAfter and BulkCopyTimeout are mutually exclusive, so the event can fire even if no rows have been sent to the database or committed.

SARowsCopied() event

This event occurs every time the number of rows specified by the NotifyAfter property have been processed.

Visual Basic syntax

```
Public Event SARowsCopied As SARowsCopiedEventHandler
```

C# syntax

```
public event SARowsCopiedEventHandler SARowsCopied;
```

Usage

The receipt of an SARowsCopied event does not imply that any rows have been sent to the database server or committed. You cannot call the Close method from this event.

SABulkCopyColumnMapping class

Defines the mapping between a column in an SABulkCopy instance's data source and a column in the instance's destination table.

Visual Basic syntax

```
Public NotInheritable Class SABulkCopyColumnMapping
```

C# syntax

```
public sealed class SABulkCopyColumnMapping
```

Remarks

The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

Custom Attribute: sealed

DestinationColumn property

Gets or sets the name of the column in the destination database table being mapped to.

Visual Basic syntax

```
Public Property DestinationColumn As String
```

C# syntax

```
public string DestinationColumn {get;set;}
```

Remarks

A string specifying the name of the column in the destination table or a null reference (Nothing in Visual Basic) if the DestinationOrdinal property has priority.

The DestinationColumn property and DestinationOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the DestinationColumn property causes the DestinationOrdinal property to be set to -1. Setting the DestinationOrdinal property causes the DestinationColumn property to be set to a null reference (Nothing in Visual Basic).

It is an error to set DestinationColumn to null or the empty string.

DestinationOrdinal property

Gets or sets the ordinal value of the column in the destination table being mapped to.

Visual Basic syntax

```
Public Property DestinationOrdinal As Integer
```

C# syntax

```
public int DestinationOrdinal {get;set;}
```

Remarks

An integer specifying the ordinal of the column being mapped to in the destination table or -1 if the property is not set.

The DestinationColumn property and DestinationOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the `DestinationColumn` property causes the `DestinationOrdinal` property to be set to -1. Setting the `DestinationOrdinal` property causes the `DestinationColumn` property to be set to a null reference (Nothing in Visual Basic).

SourceColumn property

Gets or sets the name of the column being mapped in the data source.

Visual Basic syntax

```
Public Property SourceColumn As String
```

C# syntax

```
public string SourceColumn {get;set;}
```

Remarks

A string specifying the name of the column in the data source or a null reference (Nothing in Visual Basic) if the `SourceOrdinal` property has priority.

The `SourceColumn` property and `SourceOrdinal` property are mutually exclusive. The most recently set value takes priority.

Setting the `SourceColumn` property causes the `SourceOrdinal` property to be set to -1. Setting the `SourceOrdinal` property causes the `SourceColumn` property to be set to a null reference (Nothing in Visual Basic).

It is an error to set `SourceColumn` to null or the empty string.

SourceOrdinal property

Gets or sets ordinal position of the source column within the data source.

Visual Basic syntax

```
Public Property SourceOrdinal As Integer
```

C# syntax

```
public int SourceOrdinal {get;set;}
```

Remarks

An integer specifying the ordinal of the column in the data source or -1 if the property is not set.

The `SourceColumn` property and `SourceOrdinal` property are mutually exclusive. The most recently set value takes priority.

Setting the `SourceColumn` property causes the `SourceOrdinal` property to be set to -1. Setting the `SourceOrdinal` property causes the `SourceColumn` property to be set to a null reference (Nothing in Visual Basic).

SABulkCopyColumnMappingCollection class

A collection of SABulkCopyColumnMapping objects that inherits from System.Collections.CollectionBase.

Visual Basic syntax

```
Public NotInheritable Class SABulkCopyColumnMappingCollection
Inherits System.Collections.CollectionBase
```

C# syntax

```
public sealed class SABulkCopyColumnMappingCollection :
System.Collections.CollectionBase
```

Remarks

The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

Implements: ICollection, IEnumerable, IList

Custom Attribute: sealed

DestinationOrdinalComparer class*Visual Basic syntax*

```
Private Class DestinationOrdinalComparer Implements
System.Collections.IComparer
```

C# syntax

```
private class DestinationOrdinalComparer :
System.Collections.IComparer
```

DestinationOrdinalComparer() constructor*Visual Basic syntax*

```
Public Sub New ()
```

C# syntax

```
public DestinationOrdinalComparer ()
```

Compare(object, object) method*Visual Basic syntax*

```
Public Function Compare (ByVal o1 As Object, ByVal o2 As
Object) As Integer
```

C# syntax

```
public int Compare (object o1, object o2)
```

Contains(SABulkCopyColumnMapping) method

Gets a value indicating whether a specified SABulkCopyColumnMapping object exists in the collection.

Visual Basic syntax

```
Public Function Contains (ByVal value As  
SABulkCopyColumnMapping ) As Boolean
```

C# syntax

```
public bool Contains ( SABulkCopyColumnMapping value)
```

Parameters

- **value** – A valid SABulkCopyColumnMapping object.

Returns

True if the specified mapping exists in the collection; otherwise, false.

CopyTo(SABulkCopyColumnMapping[], int) method

Copies the elements of the SABulkCopyColumnMappingCollection to an array of SABulkCopyColumnMapping items, starting at a particular index.

Visual Basic syntax

```
Public Sub CopyTo (ByVal array As SABulkCopyColumnMapping(),  
ByVal index As Integer)
```

C# syntax

```
public void CopyTo ( SABulkCopyColumnMapping[] array, int  
index)
```

Parameters

- **array** – The one-dimensional SABulkCopyColumnMapping array that is the destination of the elements copied from SABulkCopyColumnMappingCollection. The array must have zero-based indexing.
- **index** – The zero-based index in the array at which copying begins.

IndexOf(SABulkCopyColumnMapping) method

Gets or sets the index of the specified SABulkCopyColumnMapping object within the collection.

Visual Basic syntax

```
Public Function IndexOf (ByVal value As
SABulkCopyColumnMapping ) As Integer
```

C# syntax

```
public int IndexOf ( SABulkCopyColumnMapping value)
```

Parameters

- **value** – The SABulkCopyColumnMapping object to search for.

Returns

The zero-based index of the column mapping is returned, or -1 is returned if the column mapping is not found in the collection.

Remove(SABulkCopyColumnMapping) method

Removes the specified SABulkCopyColumnMapping element from the SABulkCopyColumnMappingCollection.

Visual Basic syntax

```
Public Sub Remove (ByVal value As SABulkCopyColumnMapping )
```

C# syntax

```
public void Remove ( SABulkCopyColumnMapping value)
```

Parameters

- **value** – The SABulkCopyColumnMapping object to be removed from the collection.

RemoveAt(int) method

Removes the mapping at the specified index from the collection.

Visual Basic syntax

```
Public Shadows Sub RemoveAt (ByVal index As Integer)
```

C# syntax

```
public new void RemoveAt (int index)
```

Parameters

- **index** – The zero-based index of the SABulkCopyColumnMapping object to be removed from the collection.

this property

Gets the SABulkCopyColumnMapping object at the specified index.

Visual Basic syntax

```
Public ReadOnly Property Item As SABulkCopyColumnMapping
```

C# syntax

```
public SABulkCopyColumnMapping this {get;}
```

DestinationOrdinalComparer class

Visual Basic syntax

```
Private Class DestinationOrdinalComparer Implements  
System.Collections.IComparer
```

C# syntax

```
private class DestinationOrdinalComparer :  
System.Collections.IComparer
```

DestinationOrdinalComparer() constructor

Visual Basic syntax

```
Public Sub New ()
```

C# syntax

```
public DestinationOrdinalComparer ()
```

Compare(object, object) method

Visual Basic syntax

```
Public Function Compare (ByVal o1 As Object, ByVal o2 As  
Object) As Integer
```

C# syntax

```
public int Compare (object o1, object o2)
```


SACommLinksOptionsBuilder class

Provides a simple way to create and manage the CommLinks options portion of connection strings used by the SACConnection class.

Visual Basic syntax

```
Public NotInheritable Class SACommLinksOptionsBuilder
```

C# syntax

```
public sealed class SACommLinksOptionsBuilder
```

Remarks

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

For a list of connection parameters, see Connection parameters.

Custom Attribute: sealed

GetUseLongNameAsKeyword() method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Function GetUseLongNameAsKeyword () As Boolean
```

C# syntax

```
public bool GetUseLongNameAsKeyword ()
```

Returns

True if long connection parameter names are used to build connection strings; otherwise, false.

Usage

SQL Anywhere connection parameters have both long and short forms of their names. For example, to specify the name of an ODBC data source in your connection string, you can use either of the following values: DataSourceName or DSN. By default, long connection parameter names are used to build connection strings.

SetUseLongNameAsKeyword(bool) method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Sub SetUseLongNameAsKeyword (ByVal useLongNameAsKeyword  
As Boolean)
```

C# syntax

```
public void SetUseLongNameAsKeyword (bool useLongNameAsKeyword)
```

Parameters

- **useLongNameAsKeyword** – A boolean value that indicates whether the long connection parameter name is used in the connection string.

Usage

Long connection parameter names are used by default.

ToString() method

Converts the SACCommLinksOptionsBuilder object to a string representation.

Visual Basic syntax

```
Public Overrides Function ToString () As String
```

C# syntax

```
public override string ToString ()
```

Returns

The options string being built.

All property

Gets or sets the ALL CommLinks option.

Visual Basic syntax

```
Public Property All As Boolean
```

C# syntax

```
public bool All {get;set;}
```

Remarks

Attempt to connect using the shared memory protocol first, followed by all remaining and available communication protocols. Use this setting if you are unsure of which communication protocol(s) to use.

The SACCommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

ConnectionString property

Gets or sets the connection string being built.

Visual Basic syntax

```
Public Property ConnectionString As String
```

C# syntax

```
public string ConnectionString {get;set;}
```

Remarks

The SACCommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

SharedMemory property

Gets or sets the SharedMemory protocol.

Visual Basic syntax

```
Public Property SharedMemory As Boolean
```

C# syntax

```
public bool SharedMemory {get;set;}
```

Remarks

The SACCommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

TcpOptionsBuilder property

Gets or sets an SATcpOptionsBuilder object used to create a TCP options string.

Visual Basic syntax

```
Public Property TcpOptionsBuilder As SATcpOptionsBuilder
```

C# syntax

```
public SATcpOptionsBuilder TcpOptionsBuilder {get;set;}
```

TcpOptionsString property

Gets or sets a string of TCP options.

Visual Basic syntax

```
Public Property TcpOptionsString As String
```

C# syntax

```
public string TcpOptionsString {get;set;}
```

SACommand class

A SQL statement or stored procedure that is executed against a SQL Anywhere database.

Visual Basic syntax

```
Public NotInheritable Class SACommand Inherits  
System.Data.Common.DbCommand Implements System.ICloneable
```

C# syntax

```
public sealed class SACommand: System.Data.Common.DbCommand,  
System.ICloneable
```

Remarks

Implements: IDbCommand, ICloneable

For more information, see [Accessing and manipulating data](#).

Custom Attribute: sealed

Cancel() method

Cancels the execution of an SACommand object.

Visual Basic syntax

```
Public Overrides Sub Cancel ()
```

C# syntax

```
public override void Cancel ()
```

Usage

If there is nothing to cancel, nothing happens. If there is a command in process, a "Statement interrupted by user" exception is thrown.

CreateDbParameter() method

Creates a new instance of a System.Data.Common.DbParameter object.

Visual Basic syntax

```
Protected Overrides Function CreateDbParameter () As  
DbParameter
```

C# syntax

```
protected override DbParameter CreateDbParameter ()
```

Returns

A System.Data.Common.DbParameter object.

CreateParameter() method

Provides an SAParameter object for supplying parameters to SACommand objects.

Visual Basic syntax

```
Public Shadows Function CreateParameter () As SAParameter
```

C# syntax

```
public new SAParameter CreateParameter ()
```

Returns

A new parameter, as an SAParameter object.

Usage

Stored procedures and some other SQL statements can take parameters, indicated in the text of a statement by a question mark (?).

The CreateParameter method provides an SAParameter object. You can set properties on the SAParameter to specify the value, data type, and so on for the parameter.

Dispose(bool) method

Frees the resources associated with the object.

Visual Basic syntax

```
Protected Overrides Sub Dispose (ByVal disposing As Boolean)
```

C# syntax

```
protected override void Dispose (bool disposing)
```

EndExecuteNonQuery(IAsyncResult) method

Finishes asynchronous execution of a SQL statement or stored procedure.

Visual Basic syntax

```
Public Function EndExecuteNonQuery (ByVal asyncResult As IAsyncResult ) As Integer
```

C# syntax

```
public int EndExecuteNonQuery ( IAsyncResult asyncResult)
```

Parameters

- **asyncResult** – The IAsyncResult returned by the call to SACCommand.BeginExecuteNonQuery.

Returns

The number of rows affected (the same behavior as SACCommand.ExecuteNonQuery).

Exceptions

- **ArgumentException** – The asyncResult parameter is null (Nothing in Microsoft Visual Basic).
- **InvalidOperationException** – The SACCommand.EndExecuteNonQuery(IAsyncResult) was called more than once for a single command execution, or the method was mismatched against its execution method.

Usage

You must call EndExecuteNonQuery once for every call to BeginExecuteNonQuery. The call must be after BeginExecuteNonQuery has returned. ADO.NET is not thread safe; it is your responsibility to ensure that BeginExecuteNonQuery has returned. The IAsyncResult passed to EndExecuteNonQuery must be the same as the one returned from the BeginExecuteNonQuery call that is being completed. It is an error to call EndExecuteNonQuery to end a call to BeginExecuteReader, and vice versa.

If an error occurs while executing the command, the exception is thrown when EndExecuteNonQuery is called.

There are four ways to wait for execution to complete:

(1) Call EndExecuteNonQuery.

Calling EndExecuteNonQuery blocks until the command completes. For example:

```
                SACConnection conn = new SACConnection("DSN=SQL
Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
// this will block until the command completes
int rowCount = cmd.EndExecuteNonQuery( res );
```

(2) Poll the IsCompleted property of the IAsyncResult.

You can poll the IsCompleted property of the IAsyncResult. For example:

```

        SAConnection conn = new SAConnection("DSN=SQL
Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
    // do other work
}
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );

```

(3) Use the `IAsyncResult.AsyncWaitHandle` property to get a synchronization object.

You can use the `IAsyncResult.AsyncWaitHandle` property to get a synchronization object, and wait on that. For example:

```

        SAConnection conn = new SAConnection("DSN=SQL
Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );

```

(4) Specify a callback function when calling `BeginExecuteNonQuery`.

You can specify a callback function when calling `BeginExecuteNonQuery`. For example:

```

        private void callbackFunction( IAsyncResult ar ) {
            SACCommand cmd = (SACCommand) ar.AsyncState;
            // this won't block since the command has completed
            int rowCount = cmd.EndExecuteNonQuery( ar );
        }

        // elsewhere in the code
private void DoStuff() {
    SAConnection conn = new SAConnection("DSN=SQL Anywhere 16 Demo");
    conn.Open();
    SACCommand cmd = new SACCommand(
        "UPDATE Departments"
        + " SET DepartmentName = 'Engineering'"
        + " WHERE DepartmentID=100",

```

```
        conn );
        IAsyncResult res = cmd.BeginExecuteNonQuery( callbackFunction,
cmd );
        // perform other work. The callback function will be
        // called when the command completes
    }
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

EndExecuteReader(IAsyncResult) method

Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested `SADaReader`.

Visual Basic syntax

```
Public Function EndExecuteReader (ByVal asyncResult As
IAsyncResult ) As SADaReader
```

C# syntax

```
public SADaReader EndExecuteReader ( IAsyncResult
asyncResult)
```

Parameters

- **asyncResult** – The `IAsyncResult` returned by the call to `SACommand.BeginExecuteReader`.

Returns

An `SADaReader` object that can be used to retrieve the requested rows (the same behavior as `SACommand.ExecuteReader`).

Exceptions

- **ArgumentException** – The `asyncResult` parameter is null (Nothing in Microsoft Visual Basic)
- **InvalidOperationException** – The `SACommand.EndExecuteReader(IAsyncResult)` was called more than once for a single command execution, or the method was mismatched against its execution method.

Usage

You must call `EndExecuteReader` once for every call to `BeginExecuteReader`. The call must be after `BeginExecuteReader` has returned. ADO.NET is not thread safe; it is your responsibility to ensure that `BeginExecuteReader` has returned. The `IAsyncResult` passed to `EndExecuteReader` must be the same as the one returned from the `BeginExecuteReader` call that is being completed. It is an error to call `EndExecuteReader` to end a call to `BeginExecuteNonQuery`, and vice versa.

If an error occurs while executing the command, the exception is thrown when `EndExecuteReader` is called.

There are four ways to wait for execution to complete:

(1) Call `EndExecuteReader`.

Calling `EndExecuteReader` blocks until the command completes. For example:

```

                SAConnection conn = new SAConnection("DSN=SQL
Anywhere 16 Demo");
conn.Open();
SACommand cmd = new SACommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
// this blocks until the command completes
SADataReader reader = cmd.EndExecuteReader( res );

```

(2) Poll the `IsCompleted` property of the `IAsyncResult`.

You can poll the `IsCompleted` property of the `IAsyncResult`. For example:

```

                SAConnection conn = new SAConnection("DSN=SQL
Anywhere 16 Demo");
conn.Open();
SACommand cmd = new SACommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
while( !res.IsCompleted ) {
    // do other work
}
// this does not block because the command is finished
SADataReader reader = cmd.EndExecuteReader( res );

```

(3) Use the `IAsyncResult.AsyncWaitHandle` property to get a synchronization object.

You can use the `IAsyncResult.AsyncWaitHandle` property to get a synchronization object, and wait on that. For example:

```

                SAConnection conn = new SAConnection("DSN=SQL
Anywhere 16 Demo");
conn.Open();
SACommand cmd = new SACommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this does not block because the command is finished
SADataReader reader = cmd.EndExecuteReader( res );

```

(4) Specify a callback function when calling `BeginExecuteReader`

You can specify a callback function when calling `BeginExecuteReader`. For example:

```

                private void callbackFunction( IAsyncResult ar ) {
                    SACommand cmd = (SACommand) ar.AsyncState;

```

```
// this does not block since the command has completed
SADataReader reader = cmd.EndExecuteReader();
}

// elsewhere in the code
private void DoStuff() {
    SAConnection conn = new SAConnection("DSN=SQL Anywhere 16 Demo");
    conn.Open();
    SACommand cmd = new SACommand( "SELECT * FROM Departments",
    conn );
    IAsyncResult res = cmd.BeginExecuteReader( callbackFunction,
    cmd );
    // perform other work. The callback function will be
    // called when the command completes
}
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

ExecuteDbDataReader(CommandBehavior) method

Executes the command text against the connection.

Visual Basic syntax

```
Protected Overrides Function ExecuteDbDataReader (ByVal behavior
As CommandBehavior) As DbDataReader
```

C# syntax

```
protected override DbDataReader ExecuteDbDataReader
(CommandBehavior behavior)
```

Parameters

- **behavior** – An instance of System.Data.CommandBehavior.

Returns

A System.Data.Common.DbDataReader.

ExecuteNonQuery() method

Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data definition statement.

Visual Basic syntax

```
Public Overrides Function ExecuteNonQuery () As Integer
```

C# syntax

```
public override int ExecuteNonQuery ()
```

Returns

The number of rows affected.

Usage

You can use `ExecuteNonQuery` to change the data in a database without using a `DataSet`. Do this by executing `UPDATE`, `INSERT`, or `DELETE` statements.

Although `ExecuteNonQuery` does not return any rows, output parameters or return values that are mapped to parameters are populated with data.

For `UPDATE`, `INSERT`, and `DELETE` statements, the return value is the number of rows affected by the command. For all other types of statements, and for rollbacks, the return value is -1.

ExecuteScalar() method

Executes a statement that returns a single value.

Visual Basic syntax

```
Public Overrides Function ExecuteScalar () As Object
```

C# syntax

```
public override object ExecuteScalar ()
```

Returns

The first column of the first row in the result set, or a null reference if the result set is empty.

Usage

If this method is called on a query that returns multiple rows and columns, only the first column of the first row is returned.

Prepare() method

Prepares or compiles the `SACCommand` on the data source.

Visual Basic syntax

```
Public Overrides Sub Prepare ()
```

C# syntax

```
public override void Prepare ()
```

Usage

If you call one of the `ExecuteNonQuery`, `ExecuteReader`, or `ExecuteScalar` methods after calling `Prepare`, any parameter value that is larger than the value specified by the `Size` property

is automatically truncated to the original specified size of the parameter, and no truncation errors are returned.

The truncation only happens for the following data types:

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR
- LONG NVARCHAR
- NTEXT
- BINARY
- LONG BINARY
- VARBINARY
- IMAGE

If the size property is not specified, and so is using the default value, the data is not truncated.

ResetCommandTimeout() method

Resets the CommandTimeout property to its default value of 30 seconds.

Visual Basic syntax

```
Public Sub ResetCommandTimeout ()
```

C# syntax

```
public void ResetCommandTimeout ()
```

CommandText property

Gets or sets the text of a SQL statement or stored procedure.

Visual Basic syntax

```
Public Overrides Property CommandText As String
```

C# syntax

```
public override string CommandText {get;set;}
```

Remarks

The SQL statement or the name of the stored procedure to execute. The default is an empty string.

CommandTimeout property

This feature is not supported by the SQL Anywhere .NET Data Provider.

Visual Basic syntax

```
Public Overrides Property CommandTimeout As Integer
```

C# syntax

```
public override int CommandTimeout {get;set;}
```

Remarks

To set a request timeout, use the following example.

```
cmd.CommandText = "SET OPTION request_timeout = 30";
cmd.ExecuteNonQuery();
```

CommandType property

Gets or sets the type of command represented by an SACommand.

Visual Basic syntax

```
Public Overrides Property CommandType As CommandType
```

C# syntax

```
public override CommandType CommandType {get;set;}
```

Remarks

One of the System.Data.CommandType values. The default is System.Data.CommandType.Text.

Supported command types are as follows:

- System.Data.CommandType.StoredProcedure When you specify this CommandType, the command text must be the name of a stored procedure and you must supply any arguments as SAParameter objects.
- System.Data.CommandType.Text This is the default value.

When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure. The command executes this stored procedure when you call one of the Execute methods.

Use a question mark (?) placeholder to pass parameters. For example:

```
SELECT * FROM Customers WHERE ID = ?
```

The order in which SAParameter objects are added to the SAParameterCollection must directly correspond to the position of the question mark placeholder for the parameter.

Connection property

Gets or sets the connection object to which the SACommand object applies.

Visual Basic syntax

```
Public Shadows Property Connection As SAConnection
```

C# syntax

```
public new SAConnection Connection {get;set;}
```

Remarks

The default value is a null reference. In Visual Basic it is Nothing.

DbConnection property

Gets or sets the System.Data.Common.DbConnection used by this SACommand object.

Visual Basic syntax

```
Protected Overrides Property DbConnection As DbConnection
```

C# syntax

```
protected override DbConnection DbConnection {get;set;}
```

Remarks

The connection to the data source.

DbParameterCollection property

Gets the collection of System.Data.Common.DbParameter objects.

Visual Basic syntax

```
Protected ReadOnly Overrides Property DbParameterCollection  
As DbParameterCollection
```

C# syntax

```
protected override DbParameterCollection  
DbParameterCollection {get;}
```

Remarks

The parameters of the SQL statement or stored procedure.

DbTransaction property

Gets or sets the System.Data.Common.DbTransaction within which this SACommand object executes.

Visual Basic syntax

```
Protected Overrides Property DbTransaction As DbTransaction
```

C# syntax

```
protected override DbTransaction DbTransaction {get;set;}
```

Remarks

The transaction within which a Command object of a .NET Framework data provider executes. The default value is a null reference (Nothing in Visual Basic).

DesignTimeVisible property

Gets or sets a value that indicates if the SACommand should be visible in a Windows Form Designer control.

Visual Basic syntax

```
Public Overrides Property DesignTimeVisible As Boolean
```

C# syntax

```
public override bool DesignTimeVisible {get;set;}
```

Remarks

The default is true.

True if this SACommand instance should be visible, false if this instance should not be visible. The default is false.

Parameters property

A collection of parameters for the current statement.

Visual Basic syntax

```
Public ReadOnly Shadows Property Parameters As  
SAParameterCollection
```

C# syntax

```
public new SAParameterCollection Parameters {get;}
```

Remarks

Use question marks in the CommandText to indicate parameters.

The parameters of the SQL statement or stored procedure. The default value is an empty collection.

When CommandType is set to Text, pass parameters using the question mark placeholder. For example:

```
SELECT * FROM Customers WHERE ID = ?
```

The order in which SAParameter objects are added to the SAParameterCollection must directly correspond to the position of the question mark placeholder for the parameter in the command text.

When the parameters in the collection do not match the requirements of the query to be executed, an error may result or an exception may be thrown.

Transaction property

Specifies the SATransaction object in which the SACommand executes.

Visual Basic syntax

```
Public Shadows Property Transaction As SATransaction
```

C# syntax

```
public new SATransaction Transaction {get;set;}
```

Remarks

The default value is a null reference. In Visual Basic, this is Nothing.

You cannot set the Transaction property if it is already set to a specific value and the command is executing. If you set the transaction property to an SATransaction object that is not connected to the same SAConnection object as the SACommand object, an exception will be thrown the next time you attempt to execute a statement.

For more information, see Transaction processing.

UpdatedRowSource property

Gets or sets how command results are applied to the DataRow when used by the Update method of the SADataAdapter.

Visual Basic syntax

```
Public Overrides Property UpdatedRowSource As  
UpdateRowSource
```

C# syntax

```
public override UpdateRowSource UpdatedRowSource {get;set;}
```


Remarks

One of the UpdatedRowSource values. The default value is UpdateRowSource.OutputParameters. If the command is automatically generated, this property is UpdateRowSource.None.

UpdatedRowSource.Both, which returns both resultset and output parameters, is not supported.

SACommandBuilder class

A way to generate single-table SQL statements that reconcile changes made to a DataSet with the data in the associated database.

Visual Basic syntax

```
Public NotInheritable Class SACommandBuilder Inherits
System.Data.Common.DbCommandBuilder
```

C# syntax

```
public sealed class SACommandBuilder :
System.Data.Common.DbCommandBuilder
```

Remarks

Custom Attribute: sealed

ApplyParameterInfo(DbParameter , DataRow, StatementType, bool) method

Allows the provider implementation of System.Data.Common.DbCommandBuilder to handle additional parameter properties.

Visual Basic syntax

```
Protected Overrides Sub ApplyParameterInfo (ByVal parameter As
DbParameter , ByVal row As DataRow, ByVal statementType As
StatementType, ByVal whereClause As Boolean)
```

C# syntax

```
protected override void ApplyParameterInfo ( DbParameter
parameter, DataRow row, StatementType statementType, bool
whereClause)
```

Parameters

- **parameter** – A System.Data.Common.DbParameter to which the additional modifications are applied.
- **row** – The System.Data.DataRow from the schema table provided by SADataReader.GetSchemaTable.

- **statementType** – The type of command being generated: INSERT, UPDATE or DELETE.
- **whereClause** – The value is true if the parameter is part of the UPDATE or DELETE WHERE clause, and false if it is part of the INSERT or UPDATE values.

DeriveParameters(SACommand) method

Populates the Parameters collection of the specified SACommand object.

Visual Basic syntax

```
Public Shared Sub DeriveParameters (ByVal command As  
SACommand )
```

C# syntax

```
public static void DeriveParameters ( SACommand command)
```

Parameters

- **command** – An SACommand object for which to derive parameters.

Usage

This is used for the stored procedure specified in the SACommand.

DeriveParameters overwrites any existing parameter information for the SACommand.

DeriveParameters requires an extra call to the database server. If the parameter information is known in advance, it is more efficient to populate the Parameters collection by setting the information explicitly.

GetParameterPlaceholder(int) method

Returns the placeholder for the parameter in the associated SQL statement.

Visual Basic syntax

```
Protected Overrides Function GetParameterPlaceholder (ByVal  
index As Integer) As String
```

C# syntax

```
protected override string GetParameterPlaceholder (int index)
```

Parameters

- **index** – The number to be included as part of the parameter's name.

Returns

The name of the parameter with the specified number appended.

GetSchemaTable(DbCommand) method

Returns the schema table for the SACommandBuilder object.

Visual Basic syntax

```
Protected Overrides Function GetSchemaTable (ByVal  
sourceCommand As DbCommand ) As DataTable
```

C# syntax

```
protected override DataTable GetSchemaTable ( DbCommand  
sourceCommand)
```

Parameters

- **sourceCommand** – The System.Data.Common.DbCommand for which to retrieve the corresponding schema table.

Returns

A System.Data.DataTable that represents the schema for the specific System.Data.Common.DbCommand.

InitializeCommand(DbCommand) method

Resets the System.Data.Common.DbCommand.CommandTimeout, System.Data.Common.DbCommand.Transaction, System.Data.Common.DbCommand.CommandType, and System.Data.Common.DbCommand.UpdatedRowSource properties on the System.Data.Common.DbCommand.

Visual Basic syntax

```
Protected Overrides Function InitializeCommand (ByVal command  
As DbCommand ) As DbCommand
```

C# syntax

```
protected override DbCommand InitializeCommand ( DbCommand  
command)
```

Parameters

- **command** – The System.Data.Common.DbCommand to be used by the command builder for the corresponding insert, update, or delete command.

Returns

A System.Data.Common.DbCommand instance to use for each insert, update, or delete operation. Passing a null value allows the InitializeCommand method to create a

System.Data.Common.DbCommand object based on the SELECT statement associated with the SACommandBuilder object.

QuoteIdentifier(string) method

Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.

Visual Basic syntax

```
Public Overrides Function QuoteIdentifier (ByVal unquotedIdentifier  
As String) As String
```

C# syntax

```
public override string QuoteIdentifier (string unquotedIdentifier)
```

Parameters

- **unquotedIdentifier** – The string representing the unquoted identifier that will have be quoted.

Returns

Returns a string representing the quoted form of an unquoted identifier with embedded quotes properly escaped.

SetRowUpdatingHandler(DbDataAdapter) method

Registers the SACommandBuilder object to handle the SADATAAdapter.RowUpdating event for an SADATAAdapter object.

Visual Basic syntax

```
Protected Overrides Sub SetRowUpdatingHandler (ByVal adapter As  
DbDataAdapter )
```

C# syntax

```
protected override void SetRowUpdatingHandler ( DbDataAdapter  
adapter)
```

Parameters

- **adapter** – The SADATAAdapter object to be used for the update.

UnquoteIdentifier(string) method

Returns the correct unquoted form of a quoted identifier, including properly un-escaping any embedded quotes in the identifier.

Visual Basic syntax

```
Public Overrides Function UnquoteIdentifier (ByVal quotedIdentifier
As String) As String
```

C# syntax

```
public override string UnquoteIdentifier (string quotedIdentifier)
```

Parameters

- **quotedIdentifier** – The string representing the quoted identifier that will have its embedded quotes removed.

Returns

Returns a string representing the unquoted form of a quoted identifier with embedded quotes properly un-escaped.

DataAdapter property

Specifies the SDataAdapter for which to generate statements.

Visual Basic syntax

```
Public Shadows Property DataAdapter As SDataAdapter
```

C# syntax

```
public new SDataAdapter DataAdapter {get;set;}
```

Remarks

An SDataAdapter object.

When you create a new instance of SACCommandBuilder, any existing SACCommandBuilder that is associated with this SDataAdapter is released.

SACConnectionStringBuilder class

Provides a simple way to create and manage the contents of connection strings used by the SACConnection class.

Visual Basic syntax

```
Public NotInheritable Class SACConnectionStringBuilder Inherits
SACConnectionStringBuilderBase
```

C# syntax

```
public sealed class SAConnectionStringBuilder :  
    SAConnectionStringBuilderBase
```

Remarks

The SAConnectionStringBuilder class inherits SAConnectionStringBuilderBase, which inherits DbConnectionStringBuilder.

The SAConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.

For a list of connection parameters, see Connection parameters.

Custom Attribute: sealed

ContainsKey(string) method

Determines whether the SAConnectionStringBuilder object contains a specific keyword.

Visual Basic syntax

```
Public Overrides Function ContainsKey (ByVal keyword As String)  
    As Boolean
```

C# syntax

```
public override bool ContainsKey (string keyword)
```

Parameters

- **keyword** – The keyword to locate in the SAConnectionStringBuilder.

Returns

True if the value associated with keyword has been set; otherwise, false.

Examples

The following statement determines whether the SAConnectionStringBuilder object contains the UserID keyword.

```
connectString.ContainsKey("UserID")
```

GetUseLongNameAsKeyword() method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Function GetUseLongNameAsKeyword () As Boolean
```

C# syntax

```
public bool GetUseLongNameAsKeyword ()
```

Returns

True if long connection parameter names are used to build connection strings; otherwise, false.

Usage

SQL Anywhere connection parameters have both long and short forms of their names. For example, to specify the name of an ODBC data source in your connection string, you can use either of the following values: DataSourceName or DSN. By default, long connection parameter names are used to build connection strings.

Remove(string) method

Removes the entry with the specified key from the SAConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function Remove (ByVal keyword As String) As Boolean
```

C# syntax

```
public override bool Remove (string keyword)
```

Parameters

- **keyword** – The key of the key/value pair to be removed from the connection string in this SAConnectionStringBuilder.

Returns

True if the key existed within the connection string and was removed; false if the key did not exist.

SetUseLongNameAsKeyword(bool) method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Sub SetUseLongNameAsKeyword (ByVal useLongNameAsKeyword As Boolean)
```

C# syntax

```
public void SetUseLongNameAsKeyword (bool useLongNameAsKeyword)
```

Parameters

- **useLongNameAsKeyword** – A boolean value that indicates whether the long connection parameter name is used in the connection string.

Usage

Long connection parameter names are used by default.

ShouldSerialize(string) method

Indicates whether the specified key exists in this SAConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function ShouldSerialize (ByVal keyword As String) As Boolean
```

C# syntax

```
public override bool ShouldSerialize (string keyword)
```

Parameters

- **keyword** – The key to locate in the SAConnectionStringBuilder.

Returns

True if the SAConnectionStringBuilder contains an entry with the specified key; otherwise false.

TryGetValue(string, out object) method

Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

Visual Basic syntax

```
Public Overrides Function TryGetValue (ByVal keyword As String, ByVal value As Object) As Boolean
```

C# syntax

```
public override bool TryGetValue (string keyword, out object value)
```

Parameters

- **keyword** – The key of the item to retrieve.
- **value** – The value corresponding to keyword.

Returns

true if keyword was found within the connection string; otherwise false.

AppInfo property

Gets or sets the AppInfo connection property.

Visual Basic syntax

```
Public Property AppInfo As String
```

C# syntax

```
public string AppInfo {get;set;}
```

AutoStart property

Gets or sets the AutoStart connection property.

Visual Basic syntax

```
Public Property AutoStart As String
```

C# syntax

```
public string AutoStart {get;set;}
```

AutoStop property

Gets or sets the AutoStop connection property.

Visual Basic syntax

```
Public Property AutoStop As String
```

C# syntax

```
public string AutoStop {get;set;}
```

Charset property

Gets or sets the Charset connection property.

Visual Basic syntax

```
Public Property Charset As String
```

C# syntax

```
public string Charset {get;set;}
```

CommBufferSize property

Gets or sets the CommBufferSize connection property.

Visual Basic syntax

```
Public Property CommBufferSize As Integer
```

C# syntax

```
public int CommBufferSize {get;set;}
```

CommLinks property

Gets or sets the CommLinks property.

Visual Basic syntax

```
Public Property CommLinks As String
```

C# syntax

```
public string CommLinks {get;set;}
```

Compress property

Gets or sets the Compress connection property.

Visual Basic syntax

```
Public Property Compress As String
```

C# syntax

```
public string Compress {get;set;}
```

CompressionThreshold property

Gets or sets the CompressionThreshold connection property.

Visual Basic syntax

```
Public Property CompressionThreshold As Integer
```

C# syntax

```
public int CompressionThreshold {get;set;}
```

ConnectionLifetime property

Gets or sets the ConnectionLifetime connection property.

Visual Basic syntax

```
Public Property ConnectionLifetime As Integer
```

C# syntax

```
public int ConnectionLifetime {get;set;}
```

ConnectionString property

Gets or sets the ConnectionName connection property.

Visual Basic syntax

```
Public Property ConnectionName As String
```

C# syntax

```
public string ConnectionName {get;set;}
```

ConnectionPool property

Gets or sets the ConnectionPool property.

Visual Basic syntax

```
Public Property ConnectionPool As String
```

C# syntax

```
public string ConnectionPool {get;set;}
```

ConnectionReset property

Gets or sets the ConnectionReset connection property.

Visual Basic syntax

```
Public Property ConnectionReset As Boolean
```

C# syntax

```
public bool ConnectionReset {get;set;}
```

Remarks

A DataTable that contains schema information.

ConnectionTimeout property

Gets or sets the ConnectionTimeout connection property.

Visual Basic syntax

```
Public Property ConnectionTimeout As Integer
```

C# syntax

```
public int ConnectionTimeout {get;set;}
```

The following statement displays the value of the ConnectionTimeout property.

```
MessageBox.Show( connString.ConnectionTimeout.ToString() );
```

DatabaseFile property

Gets or sets the DatabaseFile connection property.

Visual Basic syntax

```
Public Property DatabaseFile As String
```

C# syntax

```
public string DatabaseFile {get;set;}
```

DatabaseKey property

Gets or sets the DatabaseKey connection property.

Visual Basic syntax

```
Public Property DatabaseKey As String
```

C# syntax

```
public string DatabaseKey {get;set;}
```

DatabaseName property

Gets or sets the DatabaseName connection property.

Visual Basic syntax

```
Public Property DatabaseName As String
```

C# syntax

```
public string DatabaseName {get;set;}
```

DatabaseSwitches property

Gets or sets the DatabaseSwitches connection property.

Visual Basic syntax

```
Public Property DatabaseSwitches As String
```

C# syntax

```
public string DatabaseSwitches {get;set;}
```

DataSourceName property

Gets or sets the DataSourceName connection property.

Visual Basic syntax

```
Public Property DataSourceName As String
```

C# syntax

```
public string DataSourceName {get;set;}
```

DisableMultiRowFetch property

Gets or sets the DisableMultiRowFetch connection property.

Visual Basic syntax

```
Public Property DisableMultiRowFetch As String
```

C# syntax

```
public string DisableMultiRowFetch {get;set;}
```

Elevate property

Gets or sets the Elevate connection property.

Visual Basic syntax

```
Public Property Elevate As String
```

C# syntax

```
public string Elevate {get;set;}
```

EncryptedPassword property

Gets or sets the EncryptedPassword connection property.

Visual Basic syntax

```
Public Property EncryptedPassword As String
```

C# syntax

```
public string EncryptedPassword {get;set;}
```

Encryption property

Gets or sets the Encryption connection property.

Visual Basic syntax

```
Public Property Encryption As String
```

C# syntax

```
public string Encryption {get;set;}
```

Enlist property

Gets or sets the Enlist connection property.

Visual Basic syntax

```
Public Property Enlist As Boolean
```

C# syntax

```
public bool Enlist {get;set;}
```

FileDataSourceName property

Gets or sets the FileDataSourceName connection property.

Visual Basic syntax

```
Public Property FileDataSourceName As String
```

C# syntax

```
public string FileDataSourceName {get;set;}
```

ForceStart property

Gets or sets the ForceStart connection property.

Visual Basic syntax

```
Public Property ForceStart As String
```

C# syntax

```
public string ForceStart {get;set;}
```

Host property

Gets or sets the Host property.

Visual Basic syntax

```
Public Property Host As String
```

C# syntax

```
public string Host {get;set;}
```

IdleTimeout property

Gets or sets the IdleTimeout connection property.

Visual Basic syntax

```
Public Property IdleTimeout As Integer
```

C# syntax

```
public int IdleTimeout {get;set;}
```

InitString property

Gets or sets the InitString connection property.

Visual Basic syntax

```
Public Property InitString As String
```

C# syntax

```
public string InitString {get;set;}
```

Integrated property

Gets or sets the Integrated connection property.

Visual Basic syntax

```
Public Property Integrated As String
```

C# syntax

```
public string Integrated {get;set;}
```

Kerberos property

Gets or sets the Kerberos connection property.

Visual Basic syntax

```
Public Property Kerberos As String
```

C# syntax

```
public string Kerberos {get;set;}
```

Keys property

Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

Visual Basic syntax

```
Public ReadOnly Overrides Property Keys As ICollection
```

C# syntax

```
public override ICollection Keys {get;}
```

Remarks

An System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

Language property

Gets or sets the Language connection property.

Visual Basic syntax

```
Public Property Language As String
```

C# syntax

```
public string Language {get;set;}
```

LazyClose property

Gets or sets the LazyClose connection property.

Visual Basic syntax

```
Public Property LazyClose As String
```

C# syntax

```
public string LazyClose {get;set;}
```

LivenessTimeout property

Gets or sets the LivenessTimeout connection property.

Visual Basic syntax

```
Public Property LivenessTimeout As Integer
```

C# syntax

```
public int LivenessTimeout {get;set;}
```

LogFile property

Gets or sets the LogFile connection property.

Visual Basic syntax

```
Public Property LogFile As String
```

C# syntax

```
public string LogFile {get;set;}
```

MaxPoolSize property

Gets or sets the MaxPoolSize connection property.

Visual Basic syntax

```
Public Property MaxPoolSize As Integer
```


C# syntax

```
public int MaxPoolSize {get;set;}
```

MinPoolSize property

Gets or sets the MinPoolSize connection property.

Visual Basic syntax

```
Public Property MinPoolSize As Integer
```

C# syntax

```
public int MinPoolSize {get;set;}
```

NewPassword property

Gets or sets the NewPassword connection property.

Visual Basic syntax

```
Public Property NewPassword As String
```

C# syntax

```
public string NewPassword {get;set;}
```

NodeType property

Gets or sets the NodeType property.

Visual Basic syntax

```
Public Property NodeType As String
```

C# syntax

```
public string NodeType {get;set;}
```

Password property

Gets or sets the Password connection property.

Visual Basic syntax

```
Public Property Password As String
```

C# syntax

```
public string Password {get;set;}
```

PersistSecurityInfo property

Gets or sets the PersistSecurityInfo connection property.

Visual Basic syntax

```
Public Property PersistSecurityInfo As Boolean
```

C# syntax

```
public bool PersistSecurityInfo {get;set;}
```

Pooling property

Gets or sets the Pooling connection property.

Visual Basic syntax

```
Public Property Pooling As Boolean
```

C# syntax

```
public bool Pooling {get;set;}
```

PrefetchBuffer property

Gets or sets the PrefetchBuffer connection property.

Visual Basic syntax

```
Public Property PrefetchBuffer As Integer
```

C# syntax

```
public int PrefetchBuffer {get;set;}
```

PrefetchRows property

Gets or sets the PrefetchRows connection property.

Visual Basic syntax

```
Public Property PrefetchRows As Integer
```

C# syntax

```
public int PrefetchRows {get;set;}
```

Remarks

The default value is 200.

RetryConnectionTimeout property

Gets or sets the RetryConnectionTimeout property.

Visual Basic syntax

```
Public Property RetryConnectionTimeout As Integer
```

C# syntax

```
public int RetryConnectionTimeout {get;set;}
```

ServerName property

Gets or sets the ServerName connection property.

Visual Basic syntax

```
Public Property ServerName As String
```

C# syntax

```
public string ServerName {get;set;}
```

StartLine property

Gets or sets the StartLine connection property.

Visual Basic syntax

```
Public Property StartLine As String
```

C# syntax

```
public string StartLine {get;set;}
```

this property

Gets or sets the value of the connection keyword.

Visual Basic syntax

```
Public Overrides Property Item As Object
```

C# syntax

```
public override object this {get;set;}
```

Remarks

An object representing the value of the specified connection keyword.

If the keyword or type is invalid, an exception is raised. keyword is case insensitive.

When setting the value, passing NULL clears the value.

Unconditional property

Gets or sets the Unconditional connection property.

Visual Basic syntax

```
Public Property Unconditional As String
```

C# syntax

```
public string Unconditional {get;set;}
```

UserID property

Gets or sets the UserID connection property.

Visual Basic syntax

```
Public Property UserID As String
```

C# syntax

```
public string UserID {get;set;}
```

SACConnectionStringBuilderBase class

Base class of the SACConnectionStringBuilder class.

Visual Basic syntax

```
Public MustInherit Class SACConnectionStringBuilderBase  
Inherits System.Data.Common.DbConnectionStringBuilder
```

C# syntax

```
public abstract class SACConnectionStringBuilderBase :  
System.Data.Common.DbConnectionStringBuilder
```

Derived classes

- *SACConnectionStringBuilder* on page 243
- *SATcpOptionsBuilder* on page 308

Remarks

Custom Attribute: abstract

ContainsKey(string) method

Determines whether the SACConnectionStringBuilder object contains a specific keyword.

Visual Basic syntax

```
Public Overrides Function ContainsKey (ByVal keyword As String)  
As Boolean
```

C# syntax

```
public override bool ContainsKey (string keyword)
```

Parameters

- **keyword** – The keyword to locate in the SAConnectionStringBuilder.

Returns

True if the value associated with keyword has been set; otherwise, false.

Examples

The following statement determines whether the SAConnectionStringBuilder object contains the UserID keyword.

```
connectString.ContainsKey ("UserID")
```

GetUseLongNameAsKeyword() method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Function GetUseLongNameAsKeyword () As Boolean
```

C# syntax

```
public bool GetUseLongNameAsKeyword ()
```

Returns

True if long connection parameter names are used to build connection strings; otherwise, false.

Usage

SQL Anywhere connection parameters have both long and short forms of their names. For example, to specify the name of an ODBC data source in your connection string, you can use either of the following values: DataSourceName or DSN. By default, long connection parameter names are used to build connection strings.

Remove(string) method

Removes the entry with the specified key from the SAConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function Remove (ByVal keyword As String) As Boolean
```

C# syntax

```
public override bool Remove (string keyword)
```

Parameters

- **keyword** – The key of the key/value pair to be removed from the connection string in this SAConnectionStringBuilder.

Returns

True if the key existed within the connection string and was removed; false if the key did not exist.

SetUseLongNameAsKeyword(bool) method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Sub SetUseLongNameAsKeyword (ByVal useLongNameAsKeyword  
As Boolean)
```

C# syntax

```
public void SetUseLongNameAsKeyword (bool useLongNameAsKeyword)
```

Parameters

- **useLongNameAsKeyword** – A boolean value that indicates whether the long connection parameter name is used in the connection string.

Usage

Long connection parameter names are used by default.

ShouldSerialize(string) method

Indicates whether the specified key exists in this SAConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function ShouldSerialize (ByVal keyword As  
String) As Boolean
```

C# syntax

```
public override bool ShouldSerialize (string keyword)
```

Parameters

- **keyword** – The key to locate in the SAConnectionStringBuilder.

Returns

True if the SAConnectionStringBuilder contains an entry with the specified key; otherwise false.

TryGetValue(string, out object) method

Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

Visual Basic syntax

```
Public Overrides Function TryGetValue (ByVal keyword As String,
ByVal value As Object) As Boolean
```

C# syntax

```
public override bool TryGetValue (string keyword, out object
value)
```

Parameters

- **keyword** – The key of the item to retrieve.
- **value** – The value corresponding to keyword.

Returns

true if keyword was found within the connection string; otherwise false.

Keys property

Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

Visual Basic syntax

```
Public ReadOnly Overrides Property Keys As ICollection
```

C# syntax

```
public override ICollection Keys {get;}
```

Remarks

An System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

this property

Gets or sets the value of the connection keyword.

Visual Basic syntax

```
Public Overrides Property Item As Object
```

C# syntax

```
public override object this {get;set;}
```

Remarks

An object representing the value of the specified connection keyword.

If the keyword or type is invalid, an exception is raised. keyword is case insensitive.

When setting the value, passing NULL clears the value.

SADDataAdapter class

Represents a set of commands and a database connection used to fill a System.Data.DataSet and to update a database.

Visual Basic syntax

```
Public NotInheritable Class SADDataAdapter Inherits  
System.Data.Common.DbDataAdapter Implements System.ICloneable
```

C# syntax

```
public sealed class SADDataAdapter :  
System.Data.Common.DbDataAdapter, System.ICloneable
```

Remarks

The System.Data.DataSet provides a way to work with data offline. The SADDataAdapter provides methods to associate a DataSet with a set of SQL statements.

Implements: IDbDataAdapter, IDataAdapter, ICloneable

For more information, see Using the SADDataAdapter object to access and manipulate data and Accessing and manipulating data.

Custom Attribute: sealed

ClearBatch() method

Removes all SACommand objects from the batch.

Visual Basic syntax

```
Protected Overrides Sub ClearBatch ()
```

C# syntax

```
protected override void ClearBatch ()
```


CreateRowUpdatedEvent(DataRow, IDbCommand, StatementType, DataTableMapping) method

Initializes a new instance of the System.Data.Common.RowUpdatedEventArgs class.

Visual Basic syntax

```
Protected Overrides Function CreateRowUpdatedEvent (ByVal
dataRow As DataRow, ByVal command As IDbCommand, ByVal
statementType As StatementType, ByVal tableMapping As
DataTableMapping) As RowUpdatedEventArgs
```

C# syntax

```
protected override RowUpdatedEventArgs CreateRowUpdatedEvent
(DataRow dataRow, IDbCommand command, StatementType
statementType, DataTableMapping tableMapping)
```

Parameters

- **dataRow** – The System.Data.DataRow used to update the data source.
- **command** – The System.Data.IDbCommand executed during the System.Data.IDataAdapter.Update(System.Data.DataSet).
- **statementType** – Whether the command is an UPDATE, INSERT, DELETE, or SELECT statement.
- **tableMapping** – A System.Data.Common.DataTableMapping object.

Returns

A new instance of the System.Data.Common.RowUpdatedEventArgs class.

CreateRowUpdatingEvent(DataRow, IDbCommand, StatementType, DataTableMapping) method

Initializes a new instance of the System.Data.Common.RowUpdatingEventArgs class.

Visual Basic syntax

```
Protected Overrides Function CreateRowUpdatingEvent (ByVal
dataRow As DataRow, ByVal command As IDbCommand, ByVal
statementType As StatementType, ByVal tableMapping As
DataTableMapping) As RowUpdatingEventArgs
```

C# syntax

```
protected override RowUpdatingEventArgs
CreateRowUpdatingEvent (DataRow dataRow, IDbCommand command,
StatementType statementType, DataTableMapping tableMapping)
```

Parameters

- **dataRow** – The System.Data.DataRow used to update the data source.
- **command** – The System.Data.IDbCommand executed during the System.Data.IDataAdapter.Update(System.Data.DataSet).
- **statementType** – Whether the command is an UPDATE, INSERT, DELETE, or SELECT statement.
- **tableMapping** – A System.Data.Common.DataTableMapping object.

Returns

A new instance of the System.Data.Common.RowUpdatingEventArgs class.

Dispose(bool) method

Releases the unmanaged resources used by the SDataAdapter object and optionally releases the managed resources.

Visual Basic syntax

```
Protected Overrides Sub Dispose (ByVal disposing As Boolean)
```

C# syntax

```
protected override void Dispose (bool disposing)
```

Parameters

- **disposing** – True releases both managed and unmanaged resources; false releases only unmanaged resources.

GetFillParameters() method

Returns the parameters set by you when executing a SELECT statement.

Visual Basic syntax

```
Public Shadows Function GetFillParameters () As  
SAPparameter ()
```

C# syntax

```
public new SAPparameter[] GetFillParameters ()
```

Returns

An array of IDataParameter objects that contains the parameters set by the user.

InitializeBatching() method

Initializes batching for the SADataAdapter object.

Visual Basic syntax

```
Protected Overrides Sub InitializeBatching ()
```

C# syntax

```
protected override void InitializeBatching ()
```

OnRowUpdated(RowUpdatedEventArgs) method

Raises the RowUpdated event of a .NET Framework data provider.

Visual Basic syntax

```
Protected Overrides Sub OnRowUpdated (ByVal value As  
RowUpdatedEventArgs )
```

C# syntax

```
protected override void OnRowUpdated ( RowUpdatedEventArgs  
value)
```

Parameters

- **value** – A System.Data.Common.RowUpdatedEventArgs that contains the event data.

OnRowUpdating(RowUpdatingEventArgs) method

Raises the RowUpdating event of a .NET Framework data provider.

Visual Basic syntax

```
Protected Overrides Sub OnRowUpdating (ByVal value As  
RowUpdatingEventArgs )
```

C# syntax

```
protected override void OnRowUpdating ( RowUpdatingEventArgs  
value)
```

Parameters

- **value** – A System.Data.Common.RowUpdatingEventArgs that contains the event data.

TerminateBatching() method

Ends batching for the SADataAdapter object.

Visual Basic syntax

```
Protected Overrides Sub TerminateBatching ()
```

C# syntax

```
protected override void TerminateBatching ()
```

Update(DataRow[], DataTableMapping) method

Updates the tables in a database with the changes made to the DataSet.

Visual Basic syntax

```
Protected Overrides Function Update (ByVal dataRows As  
DataRow(), ByVal tableMapping As DataTableMapping) As Integer
```

C# syntax

```
protected override int Update (DataRow[] dataRows,  
DataTableMapping tableMapping)
```

Parameters

- **dataRows** – An array of System.Data.DataRow to update from.
- **tableMapping** – The System.Data.IDataAdapter.TableMappings collection to use.

Returns

The number of rows successfully updated from the System.Data.DataRow array.

Usage

The Update is carried out using the InsertCommand, UpdateCommand, and DeleteCommand on each row in the data set that has been inserted, updated, or deleted.

For more information, see Inserting, updating, and deleting rows using the SADataAdapter object.

DeleteCommand property

Specifies an SACommand object that is executed against the database when the Update method is called to delete rows in the database that correspond to deleted rows in the DataSet.

Visual Basic syntax

```
Public Shadows Property DeleteCommand As SACommand
```

C# syntax

```
public new SACommand DeleteCommand {get;set;}
```

Remarks

If this property is not set and primary key information is present in the DataSet during Update, DeleteCommand can be generated automatically by setting SelectCommand and using the SACommandBuilder. In that case, the SACommandBuilder generates any additional

commands that you do not set. This generation logic requires key column information to be present in the SelectCommand.

When DeleteCommand is assigned to an existing SACommand object, the SACommand object is not cloned. The DeleteCommand maintains a reference to the existing SACommand.

InsertCommand property

Specifies an SACommand that is executed against the database when the Update method is called that adds rows to the database to correspond to rows that were inserted in the DataSet.

Visual Basic syntax

```
Public Shadows Property InsertCommand As SACommand
```

C# syntax

```
public new SACommand InsertCommand {get;set;}
```

Remarks

The SACommandBuilder does not require key columns to generate InsertCommand.

When InsertCommand is assigned to an existing SACommand object, the SACommand is not cloned. The InsertCommand maintains a reference to the existing SACommand.

If this command returns rows, the rows may be added to the DataSet depending on how you set the UpdatedRowSource property of the SACommand object.

SelectCommand property

Specifies an SACommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.

Visual Basic syntax

```
Public Shadows Property SelectCommand As SACommand
```

C# syntax

```
public new SACommand SelectCommand {get;set;}
```

Remarks

When SelectCommand is assigned to a previously-created SACommand, the SACommand is not cloned. The SelectCommand maintains a reference to the previously-created SACommand object.

If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

The SELECT statement can also be specified in the SAdaAdapter constructor.

TableMappings property

Specifies a collection that provides the master mapping between a source table and a DataTable.

Visual Basic syntax

```
Public ReadOnly Shadows Property TableMappings As  
DataTableMappingCollection
```

C# syntax

```
public new DataTableMappingCollection TableMappings {get;}
```

Remarks

The default value is an empty collection.

When reconciling changes, the SDataAdapter uses the DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the DataSet.

The TableMappings property is not available in the .NET Compact Framework 2.0.

UpdateBatchSize property

Gets or sets the number of rows that are processed in each round-trip to the server.

Visual Basic syntax

```
Public Overrides Property UpdateBatchSize As Integer
```

C# syntax

```
public override int UpdateBatchSize {get;set;}
```

Remarks

The default value is 1.

Setting the value to something greater than 1 causes SDataAdapter.Update to execute all the insert statements in batches. The deletions and updates are executed sequentially as before, but insertions are executed afterward in batches of size equal to the value of UpdateBatchSize. Setting the value to 0 causes Update to send the insert statements in a single batch.

Setting the value to something greater than 1 causes SDataAdapter.Fill to execute all the insert statements in batches. The deletions and updates are executed sequentially as before, but insertions are executed afterward in batches of size equal to the value of UpdateBatchSize.

Setting the value to 0 causes Fill to send the insert statements in a single batch.

Setting it less than 0 is an error.

If `UpdateBatchSize` is set to something other than one, and the `InsertCommand` property is set to something that is not an INSERT statement, then an exception is thrown when calling `Fill`.

This behavior is different from `SqlDataAdapter`. It batches all types of commands.

UpdateCommand property

Specifies an `SACCommand` that is executed against the database when the `Update` method is called to update rows in the database that correspond to updated rows in the `DataSet`.

Visual Basic syntax

```
Public Shadows Property UpdateCommand As SACCommand
```

C# syntax

```
public new SACCommand UpdateCommand {get;set;}
```

Remarks

During `Update`, if this property is not set and primary key information is present in the `SelectCommand`, the `UpdateCommand` can be generated automatically if you set the `SelectCommand` property and use the `SACCommandBuilder`. Then, any additional commands that you do not set are generated by the `SACCommandBuilder`. This generation logic requires key column information to be present in the `SelectCommand`.

When `UpdateCommand` is assigned to a previously-created `SACCommand`, the `SACCommand` is not cloned. The `UpdateCommand` maintains a reference to the previously-created `SACCommand` object.

If execution of this command returns rows, these rows can be merged with the `DataSet` depending on how you set the `UpdatedRowSource` property of the `SACCommand` object.

RowUpdated() event

Occurs during an update after a command is executed against the data source.

Visual Basic syntax

```
Public Event RowUpdated As SARowUpdatedEventHandler
```

C# syntax

```
public event SARowUpdatedEventHandler RowUpdated;
```

Usage

When an attempt to update is made, the event fires.

The event handler receives an argument of type `SARowUpdatedEventArgs` containing data related to this event.

For more information, see the .NET Framework documentation for `OleDbDataAdapter.RowUpdated` Event.

RowUpdating() event

Occurs during an update before a command is executed against the data source.

Visual Basic syntax

```
Public Event RowUpdating As SARowUpdatingEventHandler
```

C# syntax

```
public event SARowUpdatingEventHandler RowUpdating;
```

Usage

When an attempt to update is made, the event fires.

The event handler receives an argument of type SARowUpdatingEventArgs containing data related to this event.

For more information, see the .NET Framework documentation for OleDbDataAdapter.RowUpdating Event.

DREnumerator class

Visual Basic syntax

```
Private NotInheritable Class DREnumerator Implements  
System.Collections.IEnumerator
```

C# syntax

```
private sealed class DREnumerator :  
System.Collections.IEnumerator
```

Remarks

Custom Attribute: sealed

DREnumerator(SDataReader) constructor

Visual Basic syntax

```
Public Sub New (ByVal dataReader As SDataReader )
```

C# syntax

```
public DREnumerator ( SDataReader dataReader)
```

MoveNext() method

Visual Basic syntax

```
Public Function MoveNext () As Boolean
```


C# syntax

```
public bool MoveNext ()
```

Reset() method

Visual Basic syntax

```
Public Sub Reset ()
```

C# syntax

```
public void Reset ()
```

Current property

Visual Basic syntax

```
Public ReadOnly Property Current As Object
```

C# syntax

```
public object Current {get;}
```

SADataSourceEnumerator class

Provides a mechanism for enumerating all available instances of SQL Anywhere database servers within the local network.

Visual Basic syntax

```
Public NotInheritable Class SADataSourceEnumerator Inherits  
System.Data.Common.DbDataSourceEnumerator
```

C# syntax

```
public sealed class SADataSourceEnumerator :  
System.Data.Common.DbDataSourceEnumerator
```

Remarks

There is no constructor for SADataSourceEnumerator.

The SADataSourceEnumerator class is not available in the .NET Compact Framework 2.0.

Custom Attribute: sealed

GetDataSources() method

Retrieves a DataTable containing information about all visible SQL Anywhere database servers.

Visual Basic syntax

```
Public Overrides Function GetDataSources () As DataTable
```

C# syntax

```
public override DataTable GetDataSources ()
```

Examples

The following code fills a DataTable with information for each database server that is available.

```
DataTable servers =  
SADataSourceEnumerator.Instance.GetDataSources ();
```

Usage

The returned table has four columns: ServerName, IPAddress, PortNumber, and DataBaseNames. There is a row in the table for each available database server.

Instance property

Gets an instance of SADataSourceEnumerator, which can be used to retrieve information about all visible SQL Anywhere database servers.

Visual Basic syntax

```
Public Shared ReadOnly Property Instance As  
SADataSourceEnumerator
```

C# syntax

```
public SADataSourceEnumerator Instance {get;}
```

SADefault class

Represents a parameter with a default value.

Visual Basic syntax

```
Public NotInheritable Class SADefault
```

C# syntax

```
public sealed class SADefault
```

Remarks

There is no constructor for SADefault.

```
SAParameter parm = new SAParameter();
parm.Value = SADefault.Value;
```

Custom Attribute: sealed

Value field

Gets the value for a default parameter.

Visual Basic syntax

```
Public Shared ReadOnly Value As SADefault
```

C# syntax

```
public static readonly SADefault Value;
```

Remarks

This field is read-only and static.

SAError class

Collects information relevant to a warning or error returned by the data source.

Visual Basic syntax

```
Public NotInheritable Class SAError
```

C# syntax

```
public sealed class SAError
```

Remarks

There is no constructor for SAError.

For information about error handling, see [Error handling](#) and the [SQL Anywhere .NET Data Provider](#).

Custom Attribute: sealed

ToString() method

The complete text of the error message.

Visual Basic syntax

```
Public Overrides Function ToString () As String
```

C# syntax

```
public override string ToString ()
```

Examples

The return value is a string in the form SAError:, followed by the Message. For example: SAError:UserId or Password not valid.

Message property

Returns a short description of the error.

Visual Basic syntax

```
Public ReadOnly Property Message As String
```

C# syntax

```
public string Message {get;}
```

NativeError property

Returns database-specific error information.

Visual Basic syntax

```
Public ReadOnly Property NativeError As Integer
```

C# syntax

```
public int NativeError {get;}
```

Source property

Returns the name of the provider that generated the error.

Visual Basic syntax

```
Public ReadOnly Property Source As String
```

C# syntax

```
public string Source {get;}
```

SqlState property

The SQL Anywhere five-character SQLSTATE following the ANSI SQL standard.

Visual Basic syntax

```
Public ReadOnly Property SqlState As String
```

C# syntax

```
public string SqlState {get;}
```

SAErrorCollection class

Collects all errors generated by the SQL Anywhere .NET Data Provider.

Visual Basic syntax

```
Public NotInheritable Class SAErrorCollection Implements
System.Collections.ICollection, System.Collections.IEnumerable
```

C# syntax

```
public sealed class SAErrorCollection :
System.Collections.ICollection, System.Collections.IEnumerable
```

Remarks

There is no constructor for SAErrorCollection. Typically, an SAErrorCollection is obtained from the SAException.Errors property.

Implements: ICollection, IEnumerable

For information about error handling, see Error handling and the SQL Anywhere .NET Data Provider.

Custom Attribute: sealed

CopyTo(Array, int) method

Copies the elements of the SAErrorCollection into an array, starting at the given index within the array.

Visual Basic syntax

```
Public Sub CopyTo (ByVal array As Array, ByVal index As Integer)
```

C# syntax

```
public void CopyTo (Array array, int index)
```

Parameters

- **array** – The array into which to copy the elements.
- **index** – The starting index of the array.

GetEnumerator() method

Returns an enumerator that iterates through the SAErrorCollection.

Visual Basic syntax

```
Public Function GetEnumerator () As IEnumerable
```

C# syntax

```
public IEnumerator GetEnumerator ()
```

Returns

An System.Collections.IEnumerator for the SAErrorCollection.

Count property

Returns the number of errors in the collection.

Visual Basic syntax

```
Public ReadOnly Property Count As Integer
```

C# syntax

```
public int Count {get;}
```

this property

Returns the error at the specified index.

Visual Basic syntax

```
Public ReadOnly Property Item As SAError
```

C# syntax

```
public SAError this {get;}
```

Remarks

An SAError object that contains the error at the specified index.

SAException class

The exception that is thrown when SQL Anywhere returns a warning or error.

Visual Basic syntax

```
Public Class SAException Inherits System.Exception
```

C# syntax

```
public class SAException : System.Exception
```

Remarks

There is no constructor for SAException. Typically, an SAException object is declared in a catch. For example:

```
...
catch( SAException ex )
{
```

```

    MessageBox.Show( ex.Errors[0].Message, "Error" );
}

```

For information about error handling, see [Error handling and the SQL Anywhere .NET Data Provider](#).

GetObjectData(SerializationInfo, StreamingContext) method

Sets the `SerializationInfo` with information about the exception.

Visual Basic syntax

```

Public Overrides Sub GetObjectData (ByVal info As
SerializationInfo, ByVal context As StreamingContext)

```

C# syntax

```

public override void GetObjectData (SerializationInfo info,
StreamingContext context)

```

Parameters

- **info** – The `SerializationInfo` that holds the serialized object data about the exception being thrown.
- **context** – The `StreamingContext` that contains contextual information about the source or destination.

Usage

Overrides `Exception.GetObjectData`.

Errors property

Returns a collection of one or more `SAError` objects.

Visual Basic syntax

```

Public ReadOnly Property Errors As SAErrorCollection

```

C# syntax

```

public SAErrorCollection Errors {get;}

```

Remarks

The `SAErrorCollection` object always contains at least one instance of the `SAError` object.

Message property

Returns the text describing the error.

Visual Basic syntax

```

Public ReadOnly Overrides Property Message As String

```

C# syntax

```
public override string Message {get;}
```

Remarks

This method returns a single string that contains a concatenation of all of the Message properties of all of the SAError objects in the Errors collection. Each message, except the last one, is followed by a carriage return.

NativeError property

Returns database-specific error information.

Visual Basic syntax

```
Public ReadOnly Property NativeError As Integer
```

C# syntax

```
public int NativeError {get;}
```

Source property

Returns the name of the provider that generated the error.

Visual Basic syntax

```
Public ReadOnly Overrides Property Source As String
```

C# syntax

```
public override string Source {get;}
```

SAFactory class

Represents a set of methods for creating instances of the iAnywhere.Data.SQLAnywhere provider's implementation of the data source classes.

Visual Basic syntax

```
Public NotInheritable Class SAFactory Inherits  
System.Data.Common.DbProviderFactory
```

C# syntax

```
public sealed class SAFactory:  
System.Data.Common.DbProviderFactory
```

Remarks

There is no constructor for SAFactory.

ADO.NET 2.0 adds two new classes, `DbProviderFactories` and `DbProviderFactory`, to make provider independent code easier to write. To use them with SQL Anywhere specify `iAnywhere.Data.SQLAnywhere` as the provider invariant name passed to `GetFactory`. For example:

```
' Visual Basic
Dim factory As DbProviderFactory =
    DbProviderFactories.GetFactory( "iAnywhere.Data.SQLAnywhere" )
Dim conn As DbConnection =
    factory.CreateConnection()

// C#
DbProviderFactory factory =
DbProviderFactories.GetFactory("iAnywhere.Data.SQLAnywhere" );
DbConnection conn = factory.CreateConnection();
```

In this example, `conn` is created as an `SAConnection` object.

For an explanation of provider factories and generic programming in ADO.NET 2.0, see *Generic Coding with the ADO.NET 2.0 Base Classes and Factories*.

The `SAFactory` class is not available in the .NET Compact Framework 2.0.

Custom Attribute: sealed

CreateCommand() method

Returns a strongly typed `System.Data.Common.DbCommand` instance.

Visual Basic syntax

```
Public Overrides Function CreateCommand () As DbCommand
```

C# syntax

```
public override DbCommand CreateCommand ()
```

Returns

A new `SACommand` object typed as `DbCommand`.

CreateCommandBuilder() method

Returns a strongly typed `System.Data.Common.DbCommandBuilder` instance.

Visual Basic syntax

```
Public Overrides Function CreateCommandBuilder () As
DbCommandBuilder
```

C# syntax

```
public override DbCommandBuilder CreateCommandBuilder ()
```

Returns

A new SACommand object typed as DbCommand.

CreateConnection() method

Returns a strongly typed System.Data.Common.DbConnection instance.

Visual Basic syntax

```
Public Overrides Function CreateConnection () As DbConnection
```

C# syntax

```
public override DbConnection CreateConnection ()
```

Returns

A new SACommand object typed as DbCommand.

CreateConnectionStringBuilder() method

Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function CreateConnectionStringBuilder () As  
DbConnectionStringBuilder
```

C# syntax

```
public override DbConnectionStringBuilder  
CreateConnectionStringBuilder ()
```

Returns

A new SACommand object typed as DbCommand.

CreateDataAdapter() method

Returns a strongly typed System.Data.Common.DbDataAdapter instance.

Visual Basic syntax

```
Public Overrides Function CreateDataAdapter () As  
DbDataAdapter
```

C# syntax

```
public override DbDataAdapter CreateDataAdapter ()
```

Returns

A new SACommand object typed as DbCommand.

CreateDataSourceEnumerator() method

Returns a strongly typed System.Data.Common.DbDataSourceEnumerator instance.

Visual Basic syntax

```
Public Overrides Function CreateDataSourceEnumerator () As
DbDataSourceEnumerator
```

C# syntax

```
public override DbDataSourceEnumerator
CreateDataSourceEnumerator ()
```

Returns

A new SACommand object typed as DbCommand.

CreateParameter() method

Returns a strongly typed System.Data.Common.DbParameter instance.

Visual Basic syntax

```
Public Overrides Function CreateParameter () As DbParameter
```

C# syntax

```
public override DbParameter CreateParameter ()
```

Returns

A new SACommand object typed as DbCommand.

CreatePermission(PermissionState) method

Returns a strongly-typed CodeAccessPermission instance.

Visual Basic syntax

```
Public Overrides Function CreatePermission (ByVal state As
PermissionState) As CodeAccessPermission
```

C# syntax

```
public override CodeAccessPermission CreatePermission
(PermissionState state)
```

Parameters

- **state** – A member of the System.Security.Permissions.PermissionState enumeration.

Returns

A new SACommand object typed as DbCommand.

CanCreateDataSourceEnumerator property

Always returns true, which indicates that an SADataSourceEnumerator object can be created.

Visual Basic syntax

```
Public ReadOnly Overrides Property  
CanCreateDataSourceEnumerator As Boolean
```

C# syntax

```
public override bool CanCreateDataSourceEnumerator {get;}
```

Remarks

A new SACommand object typed as DbCommand.

Instance field

Represents the singleton instance of the SAFactory class.

Visual Basic syntax

```
Public Shared ReadOnly Instance As SAFactory
```

C# syntax

```
public static readonly SAFactory Instance;
```

Remarks

SAFactory is a singleton class, which means only this instance of this class can exist.

Normally you would not use this field directly. Instead, you get a reference to this instance of SAFactory using System.Data.Common.DbProviderFactories.GetFactory(String). For an example, see the SAFactory description.

The SAFactory class is not available in the .NET Compact Framework 2.0.

SAInfoMessageEventArgs class

Provides data for the InfoMessage event.

Visual Basic syntax

```
Public NotInheritable Class SAInfoMessageEventArgs Inherits  
System.EventArgs
```

C# syntax

```
public sealed class SAInfoMessageEventArgs : System.EventArgs
```

Remarks

There is no constructor for SAInfoMessageEventArgs.

Custom Attribute: sealed

ToString() method

Retrieves a string representation of the InfoMessage event.

Visual Basic syntax

```
Public Overrides Function ToString () As String
```

C# syntax

```
public override string ToString ()
```

Returns

A string representing the InfoMessage event.

Errors property

Returns the collection of messages sent from the data source.

Visual Basic syntax

```
Public ReadOnly Property Errors As SAErrorCollection
```

C# syntax

```
public SAErrorCollection Errors {get;}
```

Message property

Returns the full text of the error sent from the data source.

Visual Basic syntax

```
Public ReadOnly Property Message As String
```

C# syntax

```
public string Message {get;}
```

MessageType property

Returns the type of the message.

Visual Basic syntax

```
Public ReadOnly Property MessageType As SAMessageType
```

C# syntax

```
public SAMessageType MessageType {get;}
```

Remarks

This can be one of: Action, Info, Status, or Warning.

NativeError property

Returns the SQLCODE returned by the database.

Visual Basic syntax

```
Public ReadOnly Property NativeError As Integer
```

C# syntax

```
public int NativeError {get;}
```

Source property

Returns the name of the SQL Anywhere .NET Data Provider.

Visual Basic syntax

```
Public ReadOnly Property Source As String
```

C# syntax

```
public string Source {get;}
```

SAMetaDataCollectionNames class

Provides a list of constants for use with the SAConnection.GetSchema(string) method to retrieve metadata collections.

Visual Basic syntax

```
Public NotInheritable Class SAMetaDataCollectionNames
```

C# syntax

```
public sealed class SAMetaDataCollectionNames
```

Remarks

This field is constant and read-only.

Custom Attribute: sealed

Columns field

Provides a constant for use with the SAConnection.GetSchema(string) method that represents the Columns collection.

Visual Basic syntax

```
Public Shared ReadOnly Columns As String
```

C# syntax

```
public static readonly string Columns;
```

The following code fills a `DataTable` with the `Columns` collection.

```
DataTable schema =
GetSchema( SAMetaDataCollectionNames.Columns );
```

DataSourceInformation field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `DataSourceInformation` collection.

Visual Basic syntax

```
Public Shared ReadOnly DataSourceInformation As String
```

C# syntax

```
public static readonly string DataSourceInformation;
```

The following code fills a `DataTable` with the `DataSourceInformation` collection.

```
DataTable schema =
GetSchema( SAMetaDataCollectionNames.DataSourceInformation );
```

DataTypes field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `DataTypes` collection.

Visual Basic syntax

```
Public Shared ReadOnly DataTypes As String
```

C# syntax

```
public static readonly string DataTypes;
```

The following code fills a `DataTable` with the `DataTypes` collection.

```
DataTable schema =
GetSchema( SAMetaDataCollectionNames.DataTypes );
```

ForeignKeys field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `ForeignKeys` collection.

Visual Basic syntax

```
Public Shared ReadOnly ForeignKeys As String
```

C# syntax

```
public static readonly string ForeignKeys;
```

The following code fills a `DataTable` with the `ForeignKeys` collection.

```
DataTable schema =  
GetSchema ( SAMetaDataCollectionNames.ForeignKeys );
```

IndexColumns field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `IndexColumns` collection.

Visual Basic syntax

```
Public Shared ReadOnly IndexColumns As String
```

C# syntax

```
public static readonly string IndexColumns;
```

The following code fills a `DataTable` with the `IndexColumns` collection.

```
DataTable schema =  
GetSchema ( SAMetaDataCollectionNames.IndexColumns );
```

Indexes field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `Indexes` collection.

Visual Basic syntax

```
Public Shared ReadOnly Indexes As String
```

C# syntax

```
public static readonly string Indexes;
```

The following code fills a `DataTable` with the `Indexes` collection.

```
DataTable schema =  
GetSchema ( SAMetaDataCollectionNames.Indexes );
```

MetaDataCollections field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `MetaDataCollections` collection.

Visual Basic syntax

```
Public Shared ReadOnly MetaDataCollections As String
```

C# syntax

```
public static readonly string MetaDataCollections;
```

The following code fills a `DataTable` with the `MetaDataCollections` collection.

```
DataTable schema =  
GetSchema ( SAMetaDataCollectionNames.MetaDataCollections );
```


ProcedureParameters field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `ProcedureParameters` collection.

Visual Basic syntax

```
Public Shared ReadOnly ProcedureParameters As String
```

C# syntax

```
public static readonly string ProcedureParameters;
```

The following code fills a `DataTable` with the `ProcedureParameters` collection.

```
DataTable schema =
    GetSchema ( SAMetaDataCollectionNames.ProcedureParameters );
```

Procedures field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `Procedures` collection.

Visual Basic syntax

```
Public Shared ReadOnly Procedures As String
```

C# syntax

```
public static readonly string Procedures;
```

The following code fills a `DataTable` with the `Procedures` collection.

```
DataTable schema =
    GetSchema ( SAMetaDataCollectionNames.Procedures );
```

ReservedWords field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `ReservedWords` collection.

Visual Basic syntax

```
Public Shared ReadOnly ReservedWords As String
```

C# syntax

```
public static readonly string ReservedWords;
```

The following code fills a `DataTable` with the `ReservedWords` collection.

```
DataTable schema =
    GetSchema ( SAMetaDataCollectionNames.ReservedWords );
```

Restrictions field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Restrictions collection.

Visual Basic syntax

```
Public Shared ReadOnly Restrictions As String
```

C# syntax

```
public static readonly string Restrictions;
```

The following code fills a `DataTable` with the Restrictions collection.

```
DataTable schema =  
GetSchema( SAMetaDataCollectionNames.Restrictions );
```

Tables field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Tables collection.

Visual Basic syntax

```
Public Shared ReadOnly Tables As String
```

C# syntax

```
public static readonly string Tables;
```

The following code fills a `DataTable` with the Tables collection.

```
DataTable schema =  
GetSchema( SAMetaDataCollectionNames.Tables );
```

UserDefinedTypes field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the UserDefinedTypes collection.

Visual Basic syntax

```
Public Shared ReadOnly UserDefinedTypes As String
```

C# syntax

```
public static readonly string UserDefinedTypes;
```

The following code fills a `DataTable` with the UserDefinedTypes collection.

```
DataTable schema =  
GetSchema( SAMetaDataCollectionNames.UserDefinedTypes );
```

Users field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Users collection.

Visual Basic syntax

```
Public Shared ReadOnly Users As String
```

C# syntax

```
public static readonly string Users;
```

The following code fills a `DataTable` with the Users collection.

```
DataTable schema =
GetSchema( SAMetaDataCollectionNames.Users );
```

ViewColumns field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the ViewColumns collection.

Visual Basic syntax

```
Public Shared ReadOnly ViewColumns As String
```

C# syntax

```
public static readonly string ViewColumns;
```

The following code fills a `DataTable` with the ViewColumns collection.

```
DataTable schema =
GetSchema( SAMetaDataCollectionNames.ViewColumns );
```

Views field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Views collection.

Visual Basic syntax

```
Public Shared ReadOnly Views As String
```

C# syntax

```
public static readonly string Views;
```

The following code fills a `DataTable` with the Views collection.

```
DataTable schema =
GetSchema( SAMetaDataCollectionNames.Views );
```

SAPparameter class

Represents a parameter to an SACommand, and optionally, its mapping to a DataSet column.

Visual Basic syntax

```
Public NotInheritable Class SAPparameter Inherits  
System.Data.Common.DbParameter Implements System.ICloneable
```

C# syntax

```
public sealed class SAPparameter :  
System.Data.Common.DbParameter, System.ICloneable
```

Remarks

Implements: IDbDataParameter, IDataParameter, ICloneable

Custom Attribute: sealed

ResetDbType() method

Resets the type (the values of DbType and SADBType) associated with this SAPparameter.

Visual Basic syntax

```
Public Overrides Sub ResetDbType ()
```

C# syntax

```
public override void ResetDbType ()
```

ToString() method

Returns a string containing the ParameterName.

Visual Basic syntax

```
Public Overrides Function ToString () As String
```

C# syntax

```
public override string ToString ()
```

Returns

The name of the parameter.

DbType property

Gets and sets the DbType of the parameter.

Visual Basic syntax

```
Public Overrides Property DbType As DbType
```

C# syntax

```
public override DbType DbType {get;set;}
```

Remarks

The SADBType and DbType are linked. Therefore, setting the DbType changes the SADBType to a supporting SADBType.

The value must be a member of the SADBType enumerator.

Direction property

Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Visual Basic syntax

```
Public Overrides Property Direction As ParameterDirection
```

C# syntax

```
public override ParameterDirection Direction {get;set;}
```

Remarks

One of the ParameterDirection values.

If the ParameterDirection is output, and execution of the associated SACommand does not return a value, the SAParameter contains a null value. After the last row from the last result set is read, the Output, InputOut, and ReturnValue parameters are updated.

IsNullable property

Gets and sets a value indicating whether the parameter accepts null values.

Visual Basic syntax

```
Public Overrides Property IsNullable As Boolean
```

C# syntax

```
public override bool IsNullable {get;set;}
```

Remarks

This property is true if null values are accepted; otherwise, it is false. The default is false. Null values are handled using the DBNull class.

Offset property

Gets and sets the offset to the Value property.

Visual Basic syntax

```
Public Property Offset As Integer
```

C# syntax

```
public int Offset {get;set;}
```

Remarks

The offset to the value. The default is 0.

ParameterName property

Gets and sets the name of the SAPParameter.

Visual Basic syntax

```
Public Overrides Property ParameterName As String
```

C# syntax

```
public override string ParameterName {get;set;}
```

Remarks

The default is an empty string.

The SQL Anywhere .NET Data Provider uses positional parameters that are marked with a question mark (?) instead of named parameters.

Precision property

Gets and sets the maximum number of digits used to represent the Value property.

Visual Basic syntax

```
Public Property Precision As Byte
```

C# syntax

```
public byte Precision {get;set;}
```

Remarks

The value of this property is the maximum number of digits used to represent the Value property. The default value is 0, which indicates that the data provider sets the precision for the Value property.

The Precision property is only used for decimal and numeric input parameters.

SADbType property

The SADbType of the parameter.

Visual Basic syntax

```
Public Property SADbType As SADbType
```

C# syntax

```
public SADBType SADBType {get;set;}
```

Remarks

The SADBType and DbType are linked. Therefore, setting the SADBType changes the DbType to a supporting DbType.

The value must be a member of the SADBType enumerator.

Scale property

Gets and sets the number of decimal places to which Value is resolved.

Visual Basic syntax

```
Public Property Scale As Byte
```

C# syntax

```
public byte Scale {get;set;}
```

Remarks

The number of decimal places to which Value is resolved. The default is 0.

The Scale property is only used for decimal and numeric input parameters.

Size property

Gets and sets the maximum size, in bytes, of the data within the column.

Visual Basic syntax

```
Public Overrides Property Size As Integer
```

C# syntax

```
public override int Size {get;set;}
```

Remarks

The value of this property is the maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.

The value of this property is the maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.

The Size property is used for binary and string types.

For variable length data types, the Size property describes the maximum amount of data to transmit to the server. For example, the Size property can be used to limit the amount of data sent to the server for a string value to the first one hundred bytes.

If not explicitly set, the size is inferred from the actual size of the specified parameter value. For fixed width data types, the value of Size is ignored. It can be retrieved for informational purposes, and returns the maximum amount of bytes the provider uses when transmitting the value of the parameter to the server.

SourceColumn property

Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.

Visual Basic syntax

```
Public Overrides Property SourceColumn As String
```

C# syntax

```
public override string SourceColumn {get;set;}
```

Remarks

A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

When SourceColumn is set to anything other than an empty string, the value of the parameter is retrieved from the column with the SourceColumn name. If Direction is set to Input, the value is taken from the DataSet. If Direction is set to Output, the value is taken from the data source. A Direction of InputOutput is a combination of both.

SourceColumnNullMapping property

Gets and sets value that indicates whether the source column is nullable.

Visual Basic syntax

```
Public Overrides Property SourceColumnNullMapping As Boolean
```

C# syntax

```
public override bool SourceColumnNullMapping {get;set;}
```

Remarks

This allows SACommandBuilder to generate Update statements for nullable columns correctly.

If the source column is nullable, true is returned; otherwise, false.

SourceVersion property

Gets and sets the DataRowVersion to use when loading Value.

Visual Basic syntax

```
Public Overrides Property SourceVersion As DataRowVersion
```


C# syntax

```
public override DataRowVersion SourceVersion {get;set;}
```

Remarks

Used by UpdateCommand during an Update operation to determine whether the parameter value is set to Current or Original. This allows primary keys to be updated. This property is ignored by InsertCommand and DeleteCommand. This property is set to the version of the DataRow used by the Item property, or the GetChildRows method of the DataRow object.

Value property

Gets and sets the value of the parameter.

Visual Basic syntax

```
Public Overrides Property Value As Object
```

C# syntax

```
public override object Value {get;set;}
```

Remarks

An Object that specifies the value of the parameter.

For input parameters, the value is bound to the SACommand that is sent to the server. For output and return value parameters, the value is set on completion of the SACommand and after the SADataReader is closed.

When sending a null parameter value to the server, you must specify DBNull, not null. The null value in the system is an empty object that has no value. DBNull is used to represent null values.

If the application specifies the database type, the bound value is converted to that type when the SQL Anywhere .NET Data Provider sends the data to the server. The provider attempts to convert any type of value if it supports the IConvertible interface. Conversion errors may result if the specified type is not compatible with the value.

Both the DbType and SADBType properties can be inferred by setting the Value.

The Value property is overwritten by Update.

SAParameterCollection class

Represents all parameters to an SACommand object and, optionally, their mapping to a DataSet column.

Visual Basic syntax

```
Public NotInheritable Class SAParameterCollection Inherits  
System.Data.Common.DbParameterCollection
```

C# syntax

```
public sealed class SAPParameterCollection :  
System.Data.Common.DbParameterCollection
```

Remarks

There is no constructor for SAPParameterCollection. You obtain an SAPParameterCollection object from the SACCommand.Parameters property of an SACCommand object.

Custom Attribute: sealed

SADBParametersEditor class

Visual Basic syntax

```
Private Class SADBParametersEditor Inherits CollectionEditor
```

C# syntax

```
private class SADBParametersEditor : CollectionEditor
```

SADBParametersEditor(Type) constructor

Visual Basic syntax

```
Public Sub New (ByVal type As Type)
```

C# syntax

```
public SADBParametersEditor (Type type)
```

CanSelectMultipleInstances() method

Visual Basic syntax

```
Protected Overrides Function CanSelectMultipleInstances () As  
Boolean
```

C# syntax

```
protected override bool CanSelectMultipleInstances ()
```

CreateInstance(Type) method

Visual Basic syntax

```
Protected Overrides Function CreateInstance (ByVal type As  
Type) As Object
```

C# syntax

```
protected override object CreateInstance (Type type)
```

EditValue(ITypeDescriptorContext, IServiceProvider, object) method**Visual Basic syntax**

```
Public Overrides Function EditValue (ByVal context As
ITypeDescriptorContext, ByVal provider As IServiceProvider,
ByVal value As Object) As Object
```

C# syntax

```
public override object EditValue (ITypeDescriptorContext
context, IServiceProvider provider, object value)
```

GetEditStyle(ITypeDescriptorContext) method**Visual Basic syntax**

```
Public Overrides Function GetEditStyle (ByVal context As
ITypeDescriptorContext) As UITypeEditorEditStyle
```

C# syntax

```
public override UITypeEditorEditStyle GetEditStyle
(ITypeDescriptorContext context)
```

AddWithValue(string, object) method

Adds a value to the end of this collection.

Visual Basic syntax

```
Public Function AddWithValue (ByVal parameterName As String,
ByVal value As Object) As SAParameter
```

C# syntax

```
public SAParameter AddWithValue (string parameterName, object
value)
```

Parameters

- **parameterName** – The name of the parameter.
- **value** – The value to be added.

Returns

The new SAParameter object.

Clear() method

Removes all items from the collection.

Visual Basic syntax

```
Public Overrides Sub Clear ()
```

C# syntax

```
public override void Clear ()
```

CopyTo(Array, int) method

Copies SAParameter objects from the SAParameterCollection to the specified array.

Visual Basic syntax

```
Public Overrides Sub CopyTo (ByVal array As Array, ByVal index As Integer)
```

C# syntax

```
public override void CopyTo (Array array, int index)
```

Parameters

- **array** – The array to copy the SAParameter objects into.
- **index** – The starting index of the array.

GetEnumerator() method

Returns an enumerator that iterates through the SAParameterCollection.

Visual Basic syntax

```
Public Overrides Function GetEnumerator () As IEnumerator
```

C# syntax

```
public override IEnumerator GetEnumerator ()
```

Returns

An System.Collections.IEnumerator for the SAParameterCollection object.

Insert(int, object) method

Inserts an SAParameter object in the collection at the specified index.

Visual Basic syntax

```
Public Overrides Sub Insert (ByVal index As Integer, ByVal value As Object)
```

C# syntax

```
public override void Insert (int index, object value)
```

Parameters

- **index** – The zero-based index where the parameter is to be inserted within the collection.
- **value** – The SAParameter object to add to the collection.

Remove(object) method

Removes the specified SAParameter object from the collection.

Visual Basic syntax

```
Public Overrides Sub Remove (ByVal value As Object)
```

C# syntax

```
public override void Remove (object value)
```

Parameters

- **value** – The SAParameter object to remove from the collection.

Count property

Returns the number of SAParameter objects in the collection.

Visual Basic syntax

```
Public ReadOnly Overrides Property Count As Integer
```

C# syntax

```
public override int Count {get;}
```

Remarks

The number of SAParameter objects in the collection.

IsFixedSize property

Gets a value that indicates whether the SAParameterCollection has a fixed size.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsFixedSize As Boolean
```

C# syntax

```
public override bool IsFixedSize {get;}
```

Remarks

True if this collection has a fixed size, false otherwise.

IsReadOnly property

Gets a value that indicates whether the SAParameterCollection is read-only.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsReadOnly As Boolean
```

C# syntax

```
public override bool IsReadOnly {get;}
```

Remarks

True if this collection is read-only, false otherwise.

IsSynchronized property

Gets a value that indicates whether the SAParameterCollection object is synchronized.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsSynchronized As Boolean
```

C# syntax

```
public override bool IsSynchronized {get;}
```

Remarks

True if this collection is synchronized, false otherwise.

SyncRoot property

Gets an object that can be used to synchronize access to the SAParameterCollection.

Visual Basic syntax

```
Public ReadOnly Overrides Property SyncRoot As Object
```

C# syntax

```
public override object SyncRoot {get;}
```

this property

Gets and sets the SAParameter object at the specified index.

Visual Basic syntax

```
Public Shadows Property Item As SAParameter
```

C# syntax

```
public new SAParameter this {get;set;}
```

Remarks

An SAParameter object.

In C#, this property is the indexer for the SAParameterCollection object.

An SAParameter object.

In C#, this property is the indexer for the SAParameterCollection object.

SADBParametersEditor class*Visual Basic syntax*

```
Private Class SADBParametersEditor Inherits CollectionEditor
```

C# syntax

```
private class SADBParametersEditor:CollectionEditor
```

SADBParametersEditor(Type) constructor*Visual Basic syntax*

```
Public Sub New (ByVal type As Type)
```

C# syntax

```
public SADBParametersEditor (Type type)
```

CanSelectMultipleInstances() method*Visual Basic syntax*

```
Protected Overrides Function CanSelectMultipleInstances () As Boolean
```

C# syntax

```
protected override bool CanSelectMultipleInstances ()
```

CreateInstance(Type) method*Visual Basic syntax*

```
Protected Overrides Function CreateInstance (ByVal type As Type) As Object
```

C# syntax

```
protected override object CreateInstance (Type type)
```

EditValue(ITypeDescriptorContext, IServiceProvider, object) method

Visual Basic syntax

```
Public Overrides Function EditValue (ByVal context As  
ITypeDescriptorContext, ByVal provider As IServiceProvider,  
ByVal value As Object) As Object
```

C# syntax

```
public override object EditValue (ITypeDescriptorContext  
context, IServiceProvider provider, object value)
```

GetEditStyle(ITypeDescriptorContext) method

Visual Basic syntax

```
Public Overrides Function GetEditStyle (ByVal context As  
ITypeDescriptorContext) As UITypeEditorEditStyle
```

C# syntax

```
public override UITypeEditorEditStyle GetEditStyle  
(ITypeDescriptorContext context)
```

SAPermission class

Enables the SQL Anywhere .NET Data Provider to ensure that a user has a security level adequate to access a SQL Anywhere data source.

Visual Basic syntax

```
Public NotInheritable Class SAPermission Inherits  
System.Data.Common.DBDataPermission
```

C# syntax

```
public sealed class SAPermission:  
System.Data.Common.DBDataPermission
```

Remarks

Custom Attribute: sealed

SAPermission(PermissionState) constructor

Initializes a new instance of the SAPermission class.

Visual Basic syntax

```
Public Sub New (ByVal state As PermissionState)
```


C# syntax

```
public SAPermission (PermissionState state)
```

Parameters

- **state** – One of the PermissionState values.

CreateInstance() method

Creates a new instance of an SAPermission class.

Visual Basic syntax

```
Protected Overrides Function CreateInstance () As  
DBDataPermission
```

C# syntax

```
protected override DBDataPermission CreateInstance ()
```

Returns

A new SAPermission object.

SAPermissionAttribute class

Associates a security action with a custom security attribute.

Visual Basic syntax

```
Public NotInheritable Class SAPermissionAttribute Inherits  
System.Data.Common.DBDataPermissionAttribute
```

C# syntax

```
public sealed class SAPermissionAttribute:  
System.Data.Common.DBDataPermissionAttribute
```

Remarks

Custom Attribute: sealed

SAPermissionAttribute(SecurityAction) constructor

Initializes a new instance of the SAPermissionAttribute class.

Visual Basic syntax

```
Public Sub New (ByVal action As SecurityAction)
```

C# syntax

```
public SAPermissionAttribute (SecurityAction action)
```

Parameters

- **action** – One of the SecurityAction values representing an action that can be performed using declarative security.

Returns

An SAPermissionAttribute object.

CreatePermission() method

Returns an SAPermission object that is configured according to the attribute properties.

Visual Basic syntax

```
Public Overrides Function CreatePermission () As IPermission
```

C# syntax

```
public override IPermission CreatePermission ()
```

SARowUpdatedEventArgs class

Provides data for the RowUpdated event.

Visual Basic syntax

```
Public NotInheritable Class SARowUpdatedEventArgs Inherits  
System.Data.Common.RowUpdatedEventArgs
```

C# syntax

```
public sealed class SARowUpdatedEventArgs :  
System.Data.Common.RowUpdatedEventArgs
```

Remarks

Custom Attribute: sealed

SARowUpdatedEventArgs(DataRow, IDbCommand, StatementType, DataTableMapping) constructor

Initializes a new instance of the SARowUpdatedEventArgs class.

Visual Basic syntax

```
Public Sub New (ByVal row As DataRow, ByVal command As  
IDbCommand, ByVal statementType As StatementType, ByVal  
tableMapping As DataTableMapping)
```

C# syntax

```
public SARowUpdatedEventArgs (DataRow row, IDbCommand command,  
StatementType statementType, DataTableMapping tableMapping)
```

Parameters

- **row** – The DataRow sent through an Update.
- **command** – The IDbCommand executed when Update is called.
- **statementType** – One of the StatementType values that specifies the type of query executed.
- **tableMapping** – The DataTableMapping sent through an Update.

Command property

Gets the SACCommand that is executed when DataAdapter.Update is called.

Visual Basic syntax

```
Public ReadOnly Shadows Property Command As SACCommand
```

C# syntax

```
public new SACCommand Command {get;}
```

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.

Visual Basic syntax

```
Public ReadOnly Shadows Property RecordsAffected As Integer
```

C# syntax

```
public new int RecordsAffected {get;}
```

Remarks

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

SARowUpdatingEventArgs class

Provides data for the RowUpdating event.

Visual Basic syntax

```
Public NotInheritable Class SARowUpdatingEventArgs Inherits  
System.Data.Common.RowUpdatingEventArgs
```

C# syntax

```
public sealed class SARowUpdatingEventArgs :  
System.Data.Common.RowUpdatingEventArgs
```

Remarks

Custom Attribute: sealed

SARowUpdatingEventArgs(DataRow, IDbCommand, StatementType, DataTableMapping) constructor

Initializes a new instance of the SARowUpdatingEventArgs class.

Visual Basic syntax

```
Public Sub New (ByVal row As DataRow, ByVal command As  
IDbCommand, ByVal statementType As StatementType, ByVal  
tableMapping As DataTableMapping)
```

C# syntax

```
public SARowUpdatingEventArgs (DataRow row, IDbCommand  
command, StatementType statementType, DataTableMapping  
tableMapping)
```

Parameters

- **row** – The DataRow to update.
- **command** – The IDbCommand to execute during update.
- **statementType** – One of the StatementType values that specifies the type of query executed.
- **tableMapping** – The DataTableMapping sent through an Update.

Command property

Specifies the SACommand to execute when performing the Update.

Visual Basic syntax

```
Public Shadows Property Command As SACommand
```

C# syntax

```
public new SACommand Command {get;set;}
```

SARowsCopiedEventArgs class

Represents the set of arguments passed to the SARowsCopiedEventHandler.

Visual Basic syntax

```
Public NotInheritable Class SARowsCopiedEventArgs
```

C# syntax

```
public sealed class SARowsCopiedEventArgs
```

Remarks

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

Custom Attribute: sealed

SARowsCopiedEventArgs(long) constructor

Creates a new instance of the SARowsCopiedEventArgs object.

Visual Basic syntax

```
Public Sub New (ByVal rowsCopied As Long)
```

C# syntax

```
public SARowsCopiedEventArgs (long rowsCopied)
```

Parameters

- **rowsCopied** – An 64-bit integer value that indicates the number of rows copied during the current bulk-copy operation.

Usage

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

Visual Basic syntax

```
Public Property Abort As Boolean
```

C# syntax

```
public bool Abort {get;set;}
```

Remarks

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

RowsCopied property

Gets the number of rows copied during the current bulk-copy operation.

Visual Basic syntax

```
Public ReadOnly Property RowsCopied As Long
```

C# syntax

```
public long RowsCopied {get;}
```

Remarks

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

SATcpOptionsBuilder class

Provides a simple way to create and manage the TCP options portion of connection strings used by the SAConnection object.

Visual Basic syntax

```
Public NotInheritable Class SATcpOptionsBuilder Inherits  
SAConnectionStringBuilderBase
```

C# syntax

```
public sealed class SATcpOptionsBuilder :  
SAConnectionStringBuilderBase
```

Remarks

The SATcpOptionsBuilder class is not available in the .NET Compact Framework 2.0.

Custom Attribute: sealed

ContainsKey(string) method

Determines whether the SAConnectionStringBuilder object contains a specific keyword.

Visual Basic syntax

```
Public Overrides Function ContainsKey (ByVal keyword As String)  
As Boolean
```

C# syntax

```
public override bool ContainsKey (string keyword)
```

Parameters

- **keyword** – The keyword to locate in the SAConnectionStringBuilder.

Returns

True if the value associated with keyword has been set; otherwise, false.

Examples

The following statement determines whether the SAConnectionStringBuilder object contains the UserID keyword.

```
connectString.ContainsKey("UserID")
```

GetUseLongNameAsKeyword() method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Function GetUseLongNameAsKeyword () As Boolean
```

C# syntax

```
public bool GetUseLongNameAsKeyword ()
```

Returns

True if long connection parameter names are used to build connection strings; otherwise, false.

Usage

SQL Anywhere connection parameters have both long and short forms of their names. For example, to specify the name of an ODBC data source in your connection string, you can use either of the following values: DataSourceName or DSN. By default, long connection parameter names are used to build connection strings.

Remove(string) method

Removes the entry with the specified key from the SAConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function Remove (ByVal keyword As String) As Boolean
```

C# syntax

```
public override bool Remove (string keyword)
```

Parameters

- **keyword** – The key of the key/value pair to be removed from the connection string in this SAConnectionStringBuilder.

Returns

True if the key existed within the connection string and was removed; false if the key did not exist.

SetUseLongNameAsKeyword(bool) method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

Visual Basic syntax

```
Public Sub SetUseLongNameAsKeyword (ByVal useLongNameAsKeyword  
As Boolean)
```

C# syntax

```
public void SetUseLongNameAsKeyword (bool useLongNameAsKeyword)
```

Parameters

- **useLongNameAsKeyword** – A boolean value that indicates whether the long connection parameter name is used in the connection string.

Usage

Long connection parameter names are used by default.

ShouldSerialize(string) method

Indicates whether the specified key exists in this SAConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function ShouldSerialize (ByVal keyword As  
String) As Boolean
```

C# syntax

```
public override bool ShouldSerialize (string keyword)
```

Parameters

- **keyword** – The key to locate in the SAConnectionStringBuilder.

Returns

True if the SAConnectionStringBuilder contains an entry with the specified key; otherwise false.

ToString() method

Converts the SATcpOptionsBuilder object to a string representation.

Visual Basic syntax

```
Public Overrides Function ToString () As String
```


C# syntax

```
public override string ToString ()
```

Returns

The options string being built.

TryGetValue(string, out object) method

Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

Visual Basic syntax

```
Public Overrides Function TryGetValue (ByVal keyword As String,  
ByVal value As Object) As Boolean
```

C# syntax

```
public override bool TryGetValue (string keyword, out object  
value)
```

Parameters

- **keyword** – The key of the item to retrieve.
- **value** – The value corresponding to keyword.

Returns

true if keyword was found within the connection string; otherwise false.

Broadcast property

Gets or sets the Broadcast option.

Visual Basic syntax

```
Public Property Broadcast As String
```

C# syntax

```
public string Broadcast {get;set;}
```

BroadcastListener property

Gets or sets the BroadcastListener option.

Visual Basic syntax

```
Public Property BroadcastListener As String
```

C# syntax

```
public string BroadcastListener {get;set;}
```

ClientPort property

Gets or sets the ClientPort option.

Visual Basic syntax

```
Public Property ClientPort As String
```

C# syntax

```
public string ClientPort {get;set;}
```

DoBroadcast property

Gets or sets the DoBroadcast option.

Visual Basic syntax

```
Public Property DoBroadcast As String
```

C# syntax

```
public string DoBroadcast {get;set;}
```

Host property

Gets or sets the Host option.

Visual Basic syntax

```
Public Property Host As String
```

C# syntax

```
public string Host {get;set;}
```

IPV6 property

Gets or sets the IPV6 option.

Visual Basic syntax

```
Public Property IPV6 As String
```

C# syntax

```
public string IPV6 {get;set;}
```

Keys property

Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

Visual Basic syntax

```
Public ReadOnly Overrides Property Keys As ICollection
```

C# syntax

```
public override ICollection Keys {get;}
```

Remarks

An System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

LDAP property

Gets or sets the LDAP option.

Visual Basic syntax

```
Public Property LDAP As String
```

C# syntax

```
public string LDAP {get;set;}
```

LocalOnly property

Gets or sets the LocalOnly option.

Visual Basic syntax

```
Public Property LocalOnly As String
```

C# syntax

```
public string LocalOnly {get;set;}
```

MyIP property

Gets or sets the MyIP option.

Visual Basic syntax

```
Public Property MyIP As String
```

C# syntax

```
public string MyIP {get;set;}
```

ReceiveBufferSize property

Gets or sets the ReceiveBufferSize option.

Visual Basic syntax

```
Public Property ReceiveBufferSize As Integer
```

C# syntax

```
public int ReceiveBufferSize {get;set;}
```

SendBufferSize property

Gets or sets the Send BufferSize option.

Visual Basic syntax

```
Public Property SendBufferSize As Integer
```

C# syntax

```
public int SendBufferSize {get;set;}
```

ServerPort property

Gets or sets the ServerPort option.

Visual Basic syntax

```
Public Property ServerPort As String
```

C# syntax

```
public string ServerPort {get;set;}
```

TDS property

Gets or sets the TDS option.

Visual Basic syntax

```
Public Property TDS As String
```

C# syntax

```
public string TDS {get;set;}
```

this property

Gets or sets the value of the connection keyword.

Visual Basic syntax

```
Public Overrides Property Item As Object
```

C# syntax

```
public override object this {get;set;}
```

Remarks

An object representing the value of the specified connection keyword.

If the keyword or type is invalid, an exception is raised. keyword is case insensitive.

When setting the value, passing NULL clears the value.

Timeout property

Gets or sets the Timeout option.

Visual Basic syntax

```
Public Property Timeout As Integer
```

C# syntax

```
public int Timeout {get;set;}
```

VerifyServerName property

Gets or sets the VerifyServerName option.

Visual Basic syntax

```
Public Property VerifyServerName As String
```

C# syntax

```
public string VerifyServerName {get;set;}
```

SATransaction class

Represents a SQL transaction.

Visual Basic syntax

```
Public NotInheritable Class SATransaction Inherits  
System.Data.Common.DbTransaction
```

C# syntax

```
public sealed class SATransaction:  
System.Data.Common.DbTransaction
```

Remarks

There is no constructor for SATransaction. To obtain an SATransaction object, use one of the BeginTransaction methods. To associate a command with a transaction, use the SACommand.Transaction property.

For more information, see Transaction processing and Inserting, updating, and deleting rows using the SACommand object.

Custom Attribute: sealed

Commit() method

Commits the database transaction.

Visual Basic syntax

```
Public Overrides Sub Commit ()
```

C# syntax

```
public override void Commit ()
```

Save(string) method

Creates a savepoint in the transaction that can be used to roll back a portion of the transaction, and specifies the savepoint name.

Visual Basic syntax

```
Public Sub Save (ByVal savePoint As String)
```

C# syntax

```
public void Save (string savePoint)
```

Parameters

- **savePoint** – The name of the savepoint to which to roll back.

Connection property

The SAConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Visual Basic syntax

```
Public ReadOnly Shadows Property Connection As SAConnection
```

C# syntax

```
public new SAConnection Connection {get;}
```

Remarks

A single application can have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by BeginTransaction.

DbConnection property

Specifies the System.Data.Common.DbConnection object associated with the transaction.

Visual Basic syntax

```
Protected ReadOnly Overrides Property DbConnection As  
DbConnection
```

C# syntax

```
protected override DbConnection DbConnection {get;}
```

Remarks

The System.Data.Common.DbConnection object associated with the transaction.

IsolationLevel property

Specifies the isolation level for this transaction.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsolationLevel As System.Data.IsolationLevel
```

C# syntax

```
public override System.Data.IsolationLevel IsolationLevel {get;}
```

Remarks

The IsolationLevel for this transaction. This can be one of:

- Unspecified
- Chaos
- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serializable
- Snapshot

The default is ReadCommitted.

SAIsolationLevel property

Specifies the extended isolation level for this transaction.

Visual Basic syntax

```
Public ReadOnly Property SAIsolationLevel As SAIsolationLevel
```

C# syntax

```
public SAIsolationLevel SAIsolationLevel {get;}
```

Remarks

The SAIsolationLevel for this transaction. This can be one of:

.NET Application Programming

- Unspecified
- Chaos
- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serializable
- Snapshot
- StatementSnapshot
- ReadOnlySnapshot

The default is ReadCommitted.

Parallel transactions are not supported. Therefore, the SAIsolationLevel applies to the entire transaction.

OLE DB and ADO Development

SAP Sybase IQ includes an OLE DB provider for OLE DB and ADO.

OLE DB is a set of Component Object Model (COM) interfaces developed by Microsoft, which provide applications with uniform access to data stored in diverse information sources and that also provide the ability to implement additional database services. These interfaces support the amount of DBMS functionality appropriate to the data store, enabling it to share its data.

ADO is an object model for programmatically accessing, editing, and updating a wide variety of data sources through OLE DB system interfaces. ADO is also developed by Microsoft. Most developers using the OLE DB programming interface do so by writing to the ADO API rather than directly to the OLE DB API.

Do not confuse the ADO interface with ADO.NET. ADO.NET is a separate interface.

Refer to the Microsoft Developer Network for documentation on OLE DB and ADO programming. For SAP Sybase IQ-specific information about OLE DB and ADO development, use this document.

OLE DB

OLE DB is a data access model from Microsoft. It uses the Component Object Model (COM) interfaces and, unlike ODBC, OLE DB does not assume that the data source uses a SQL query processor.

SAP Sybase IQ includes an *OLE DB provider* named SAOLEDB. This provider is available for current Windows platforms. The provider is not available for Windows Mobile platforms.

You can also access SAP Sybase IQ using the Microsoft OLE DB Provider for ODBC (MSDASQL), together with the SQL Anywhere ODBC driver.

Using the SAP Sybase IQ OLE DB provider brings several benefits:

- Some features, such as updating through a cursor, are not available using the OLE DB/ODBC bridge.
- If you use the SAP Sybase IQ OLE DB provider, ODBC is not required in your deployment.
- MSDASQL allows OLE DB clients to work with any ODBC driver, but does not guarantee that you can use the full range of functionality of each ODBC driver. Using the SAP Sybase IQ provider, you can get full access to SAP Sybase IQ features from OLE DB programming environments.

Connecting Using OLE DB

SAP Sybase IQ includes an OLE DB provider as an alternative to ODBC. OLE DB is a data access model from Microsoft that uses the Component Object Model (COM) interfaces.

Unlike ODBC, OLE DB does not assume that the data source uses a SQL query processor. Although OLE DB requires a Windows client, you can use OLE DB to access Windows and UNIX servers.

SAP Sybase IQ OLE DB support differs from SQL Anywhere support. SAP Sybase IQ supports Dynamic (dynamic scroll), Static (insensitive) and Forward only (no-scroll) cursors, but does not support Keyset (scroll) cursors. In SAP Sybase IQ the isolation level is always 3, no matter what you specify.

SAP Sybase IQ supports Dynamic (dynamic scroll), Static (insensitive) and Forward only (no-scroll) cursors, but does not support Keyset (scroll) cursors. In SAP Sybase IQ the isolation level is always 3, no matter what you specify.

SAP Sybase IQ does not support Windows CE or remote updates through a cursor.

Additional Information

Programming > OLE DB and ADO Development > OLE DB Connection Parameters

Supported Platforms

The SAP Sybase IQ OLE DB provider is designed to work with Microsoft Data Access Components (MDAC) 2.8 and later versions.

Distributed Transactions in OLE DB

The OLE DB driver can be used as a resource manager in a distributed transaction environment.

ADO Programming with SAP Sybase IQ

ADO (ActiveX Data Objects) is a data access object model exposed through an Automation interface, which allows client applications to discover the methods and properties of objects at runtime without any prior knowledge of the object. Automation allows scripting languages like Visual Basic to use a standard data access object model. ADO uses OLE DB to provide data access.

Using the SAP Sybase IQ OLE DB provider, you get full access to SAP Sybase IQ features from an ADO programming environment.

This section describes how to perform basic tasks while using ADO from Visual Basic. It is not a complete guide to programming using ADO.

Code samples from this section can be found in the %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\VBSampler\vbsampler.sln project file.

For information about programming in ADO, see your development tool documentation.

How to Connect to a Database Using the Connection Object

This section describes a simple Visual Basic routine that connects to a database.

Sample Code

You can try this routine by placing a command button named cmdTestConnection on a form, and pasting the routine into its Click event. Run the program and click the button to connect and then disconnect.

```
Private Sub cmdTestConnection_Click(
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdTestConnection.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer

    On Error GoTo HandleError

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString =
        "Data Source=Sybase IQ Demo"
    myConn.Open()
    MsgBox("Connection succeeded")
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

Notes

The sample carries out the following tasks:

- It declares the variables used in the routine.
- It establishes a connection, using the SAP Sybase IQ OLE DB provider, to the sample database.
- It uses a Command object to execute a simple statement, which displays a message in the database server messages window.
- It closes the connection.

How to Execute Statements Using the Command Object

This section describes a simple routine that sends a simple SQL statement to the database.

Sample Code

You can try this routine by placing a command button named cmdUpdate on a form, and pasting the routine into its Click event. Run the program and click the button to connect, display a message in the database server messages window, and then disconnect.

```
Private Sub cmdUpdate_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdUpdate.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer

    On Error GoTo HandleError

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()

    'Execute a command
    myCommand.CommandText = _
        "UPDATE Customers SET GivenName='Liz' WHERE ID=102"
    myCommand.ActiveConnection = myConn
    myCommand.Execute(cAffected)
    MsgBox(CStr(cAffected) & " rows affected.", _
        MsgBoxStyle.Information)

    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

Notes

After establishing a connection, the example code creates a Command object, sets its CommandText property to an update statement, and sets its ActiveConnection property to the current connection. It then executes the update statement and displays the number of rows affected by the update in a window.

In this example, the update is sent to the database and committed when it is executed.

You can also perform updates through a cursor.

How to Obtain Result Sets Using the Recordset Object

The ADO Recordset object represents the result set of a query. You can use it to view data from a database.

Sample code

You can try this routine by placing a command button named cmdQuery on a form and pasting the routine into its Click event. Run the program and click the button to connect, display a message in the database server messages window, execute a query and display the first few rows in windows, and then disconnect.

```
Private Sub cmdQuery_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdQuery.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim myRS As New ADODB.Recordset

    On Error GoTo ErrorHandler

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.CursorLocation = _
        ADODB.CursorLocationEnum.adUseServer
    myConn.Mode = _
        ADODB.ConnectModeEnum.adModeReadWrite
    myConn.IsolationLevel = _
        ADODB.IsolationLevelEnum.adXactCursorStability
    myConn.Open()

    'Execute a query
    myRS = New ADODB.Recordset
    myRS.CacheSize = 50
    myRS.Provider = "SAOLEDB"
    myRS.ActiveConnection = myConn
    myRS.CursorType = ADODB.CursorTypeEnum.adOpenKeyset
    myRS.LockType = ADODB.LockTypeEnum.adLockOptimistic
    myRS.Open()

    'Scroll through the first few results
    myRS.MoveFirst()
    For i = 1 To 5
        MsgBox(myRS.Fields("CompanyName").Value, _
            MsgBoxStyle.Information)
        myRS.MoveNext()
    Next
```

```

myRS.Close()
myConn.Close()
Exit Sub

ErrorHandler:
MsgBox (ErrorToString (Err.Number))
Exit Sub
End Sub

```

Notes

The Recordset object in this example holds the results from a query on the Customers table. The For loop scrolls through the first several rows and displays the CompanyName value for each row.

This is a simple example of using a cursor from ADO.

The Recordset Object

When working with SAP Sybase IQ, the ADO Recordset represents a cursor. You can choose the type of cursor by declaring a CursorType property of the Recordset object before you open the Recordset. The choice of cursor type controls the actions you can take on the Recordset and has performance implications.

Cursor types

ADO has its own naming convention for cursor types.

The available cursor types, the corresponding cursor type constants, and the SQL Anywhere types they are equivalent to, are as follows:

ADO cursor type	ADO constant	SAP Sybase IQ type
Dynamic cursor	adOpenDynamic	Dynamic scroll cursor
Keyset cursor	adOpenKeyset	Scroll cursor
Static cursor	adOpenStatic	Insensitive cursor
Forward only	adOpenForwardOnly	No-scroll cursor

Sample code

The following code sets the cursor type for an ADO Recordset object:

```

Dim myRS As New ADODB.Recordset
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic

```

Row Updates Through a Cursor Using the Recordset Object

The SAP Sybase IQ OLE DB provider lets you update a result set through a cursor. This capability is not available through the MSDASQL provider.

Updating Record Sets

You can update the database through a Recordset.

```
Private Sub cmdUpdateThroughCursor_Click(
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdUpdateThroughCursor.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim SQLString As String

    On Error GoTo HandleError

    ' Connect
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()
    myConn.BeginTrans()
    SQLString = "SELECT * FROM Customers"
    myRS.Open(SQLString, myConn, _
        ADODB.CursorTypeEnum.adOpenDynamic, _
        ADODB.LockTypeEnum.adLockBatchOptimistic)

    If myRS.EOF And myRS.BOF Then
        MsgBox("Recordset is empty!", 16, "Empty Recordset")
    Else
        MsgBox("Cursor type: " & CStr(myRS.CursorType), _
            MsgBoxStyle.Information)
        myRS.MoveFirst()
        For i = 1 To 3
            MsgBox("Row: " & CStr(myRS.Fields("ID").Value), _
                MsgBoxStyle.Information)
            If i = 2 Then
                myRS.Update("City", "Toronto")
                myRS.UpdateBatch()
            End If
            myRS.MoveNext()
        Next i
        myRS.Close()
    End If
    myConn.CommitTrans()
    myConn.Close()
    Exit Sub

HandleError:
```

```

MsgBox (ErrorToString (Err.Number))
Exit Sub

End Sub

```

Notes

If you use the `adLockBatchOptimistic` setting on the Recordset, the `myRS.Update` method does not make any changes to the database itself. Instead, it updates a local copy of the Recordset.

The `myRS.UpdateBatch` method makes the update to the database server, but does not commit it, because it is inside a transaction. If an `UpdateBatch` method was invoked outside a transaction, the change would be committed.

The `myConn.CommitTrans` method commits the changes. The Recordset object has been closed by this time, so there is no issue of whether the local copy of the data is changed or not.

ADO Transactions

By default, any change you make to the database using ADO is committed when it is executed. This includes explicit updates, and the `UpdateBatch` method on a Recordset. However, the previous section illustrated that you can use the `BeginTrans` and `RollbackTrans` or `CommitTrans` methods on the Connection object to use transactions.

The transaction isolation level is set as a property of the Connection object. The `IsolationLevel` property can take on one of the following values:

ADO isolation level	Constant	SAP Sybase IQ level
Unspecified	<code>adXactUnspecified</code>	Not applicable. Set to 0
Chaos	<code>adXactChaos</code>	Unsupported. Set to 0
Browse	<code>adXactBrowse</code>	0
Read uncommitted	<code>adXactReadUncommitted</code>	0
Cursor stability	<code>adXactCursorStability</code>	1
Read committed	<code>adXactReadCommitted</code>	1
Repeatable read	<code>adXactRepeatableRead</code>	2
Isolated	<code>adXactIsolated</code>	3
Serializable	<code>adXactSerializable</code>	3
Snapshot	2097152	4
Statement snapshot	4194304	5
Readonly statement snapshot	8388608	6

OLE DB Connection Parameters

OLE DB connection parameters are defined by Microsoft. The SAP Sybase IQ OLE DB provider supports a subset of these connection parameters. A typical connection string looks like this:

```
"Provider=SAOLEDB;Data Source=myDsn;Initial Catalog=myDbn;
  User ID=myUid;Password=myPwd"
```

Below are the OLE DB connection parameters that are supported by the provider. In some cases, OLE DB connection parameters are identical to (for example, Password) or resemble (for example, User ID) SAP Sybase IQ connection parameters. Note the use of spaces in many of these connection parameters.

- **Provider** – This parameter is used to identify the SQL Anywhere OLE DB provider (SAOLEDB).
- **User ID** – This connection parameter maps directly to the SAP Sybase IQ UserID (UID) connection parameter.
- **Password** – This connection parameter maps directly to the SAP Sybase IQ Password (PWD) connection parameter.
- **Data Source** – This connection parameter maps directly to the SAP Sybase IQ DataSourceName (DSN) connection parameter. For example: `Data Source=Sybase IQ Demo`.
- **Initial Catalog** – This connection parameter maps directly to the SAP Sybase IQ DatabaseName (DBN) connection parameter. For example: `Initial Catalog=demo`.
- **Location** – This connection parameter maps directly to the SAP Sybase IQ Host connection parameter. The parameter value has the same form as the Host parameter value. For example: `Location=localhost:4444`.
- **Extended Properties** – This connection parameter is used by OLE DB to pass in all the SAP Sybase IQ specific connection parameters. For example: `Extended Properties="UserID=DBA;DBKEY=V3moj3952B;DBF=demo.db"`.

ADO uses this connection parameter to collect and pass in all the connection parameters that it does not recognize.

Some Microsoft connection windows have a field called **Prov String** or **Provider String**. The contents of this field are passed as the value to Extended Properties.

- **OLE DB Services** – This connection parameter is not directly handled by the SAP Sybase IQ OLE DB provider. It controls connection pooling in ADO.
- **Prompt** – This connection parameter governs how a connection attempt handles errors. The possible prompt values are 1, 2, 3, or 4. The meanings are DBPROMPT_PROMPT (1), DBPROMPT_COMPLETE (2), DBPROMPT_COMPLETEREQUIRED (3), and DBPROMPT_NOPROMPT (4).

The default prompt value is 4 which means the provider does not present a connect window. Setting the prompt value to 1 causes a connect window to always appear. Setting the prompt value to 2 causes a connect window to appear if the initial connection attempt fails. Setting the prompt value to 3 causes a connect window to appear if the initial connection attempt fails but the provider disables the controls for any information not required to connect to the data source.

- **Window Handle** – The application can pass the handle of the parent window, if applicable, or a null pointer if either the window handle is not applicable or the provider does present any windows. The window handle value is typically 0 (NULL).

Other OLE DB connection parameters can be specified but they are ignored by the OLE DB provider.

When the SAP Sybase IQ OLE DB provider is invoked, it gets the property values for the OLE DB connection parameters. Here is a typical set of property values obtained from Microsoft's RowsetViewer application.

```
User ID '<user_id>'
Password '<password>'
Location 'localhost:4444'
Initial Catalog 'demo'
Data Source 'testds'
Extended Properties 'appinfo=api=oledb'
Prompt 2
Window Handle 0
```

The connection string that the provider constructs from this set of parameter values is:

```
'DSN=testds;HOST=localhost:
4444;DBN=demo;UID=<user_id>;PWD=<password>;appinfo=api=oledb'
```

The SAP Sybase IQ OLE DB provider uses the connection string, Window Handle, and Prompt values as parameters to the database server connection call that it makes.

This is a simple ADO connection string example.

```
connection.Open
"Provider=SAOLEDB;UserID=<user_id>;Location=localhost:
4444;Pwd=<password>"
```

ADO parses the connection string and passes all of the unrecognized connection parameters in Extended Properties. When the SAP Sybase IQ OLE DB provider is invoked, it gets the property values for the OLE DB connection parameters. Here is the set of property values obtained from the ADO application that used the connection string shown above.

```
User ID ''
Password ''
Location 'localhost:4444'
Initial Catalog ''
Data Source ''
Extended Properties 'UserID=<user_id>;Pwd=<password>'
Prompt 4
Window Handle 0
```

The connection string that the provider constructs from this set of parameter values is:

```
'HOST=localhost:4444; UserID=<user_id>;Pwd=<password>'
```

The provider uses the connection string, Window Handle, and Prompt values as parameters to the database server connection call that it makes.

OLE DB Connection Pooling

The .NET Framework Data Provider for OLE DB automatically pools connections using OLE DB session pooling.

When the application closes the connection, it is not actually closed. Instead, the connection is held for a period of time. When your application re-opens a connection, ADO/OLE DB recognizes that the application is using an identical connection string and reuses the open connection. For example, if the application does an Open/Execute/Close 100 times, there is only 1 actual open and 1 actual close. The final close occurs after about 1 minute of idle time.

If a connection is terminated by external means (such as a forced disconnect using an administrative tool), ADO/OLE DB does not know that this has occurred until the next interaction with the server. Caution should be exercised before resorting to forcible disconnects.

The flag that controls connection pooling is `DBPROPVAL_OS_RESOURCEPOOLING` (1). This flag can be turned off using a connection parameter in the connection string.

If you specify `OLE DB Services=-2` in your connection string, then connection pooling is disabled. Here is a sample connection string:

```
Provider=SAOLEDB;OLE DB Services=-2;...
```

If you specify `OLE DB Services=-4` in your connection string, then connection pooling and transaction enlistment are disabled. Here is a sample connection string:

```
Provider=SAOLEDB;OLE DB Services=-4;...
```

If you disable connection pooling, there is a performance penalty if your application frequently opens and closes connections using the same connection string.

Microsoft Linked Servers

A Microsoft Linked Server can be created that uses the SAP Sybase IQ OLE DB provider to obtain access to an SAP Sybase IQ database. SQL queries can be issued using either the Microsoft four-part table referencing syntax or the Microsoft `OPENQUERY` SQL function. An example of the four-part syntax follows.

```
SELECT * FROM SADATABASE.demo.GROUPO.Customers
```

In this example, `SADATABASE` is the name of the Linked Server, `demo` is the catalog or database name, `GROUPO` is the table owner in the SAP Sybase IQ database, and `Customers` is the table name in the SAP Sybase IQ database.

The other form uses the Microsoft OPENQUERY function.

```
SELECT * FROM OPENQUERY( SADBATABASE, 'SELECT * FROM Customers' )
```

In the OPENQUERY syntax, the second SELECT statement ('SELECT * FROM Customers') is passed to the SAP Sybase IQ server for execution.

For complex queries, OPENQUERY may be the better choice since the entire query is evaluated on the SAP Sybase IQ server. With the four-part syntax, SQL Server may retrieve the contents of all tables referenced by the query before it can evaluate it (for example, queries with WHERE, JOIN, nested queries, etc.). For queries involving very large tables, processing time may be very poor when using four-part syntax. In the following four-part query example, SQL Server passes a simple SELECT on the entire table (no WHERE clause) to the SAP Sybase IQ database server via the OLE DB provider and then evaluates the WHERE condition itself.

```
SELECT ID, Surname, GivenName FROM [SADBATABASE].[demo].[GROUPO].  
[Customers]  
WHERE Surname = 'Elkins'
```

Instead of returning one row in the result set to SQL Server, all rows are returned and then this result set is reduced to one row by SQL Server. The following example produces an identical result but only one row is returned to SQL Server.

```
SELECT * FROM OPENQUERY( SADBATABASE,  
    'SELECT ID, Surname, GivenName FROM [GROUPO].[Customers]  
    WHERE Surname = ''Elkins'' )
```

You can set up a Linked Server that uses the SAP Sybase IQ OLE DB provider using a Microsoft SQL Server interactive application or a SQL Server script.

Note: Before setting up a Linked Server, there are a few things to consider when using Windows Vista or later versions of Windows. SQL Server runs as a service on your system. Depending on how the service is set up on Windows Vista or later versions, a service may not be able to use shared memory connections, it may not be able to start a server, and it may not be able to access User Data Source definitions. For example, a service logged in as a **Network Service** cannot start servers, connect via shared memory, or access User Data Sources. For these situations, the SAP Sybase IQ server must be started ahead of time and the TCP/IP communication protocol must be used. Also, if a data source is to be used, it must be a System Data Source.

Setting up a Linked Server Using an Interactive Application

Use a Microsoft SQL Server interactive application to create a Microsoft Linked Server that uses the SAP Sybase IQ OLE DB provider to obtain access to an SAP Sybase IQ database.

Prerequisites

SQL Server 2000 or later.

Task

1. For Microsoft SQL Server 2005/2008, start SQL Server Management Studio. For other versions of SQL Server, the name of this application and the steps to setting up a Linked Server may vary.

In the **Object Explorer** pane, expand **Server Objects » Linked Servers**. Right-click **Linked Servers** and then click **New Linked Server**.

2. Fill in the **General** page.

The **Linked Server** field on the **General** page should contain a **Linked Server** name (like SADATABASE in the example above).

The **Other Data Source** option should be chosen, and **SQL Anywhere OLE DB Provider 16** should be chosen from the **Provider** list.

The **Product Name** field can be anything you like (for example, SAP Sybase IQ or your application name).

The **Data Source** field can contain an ODBC data source name (DSN). This is a convenience option and a data source name is not required. If you use a System DSN, it must be a 32-bit DSN for 32-bit versions of SQL Server or a 64-bit DSN for 64-bit versions of SQL Server.

```
Data Source: SAP Sybase IQ 16 Demo
```

The **Provider String** field can contain additional connection parameters such as UserID (UID), ServerName (Server), and DatabaseFile (DBF).

```
Provider string: Server=myserver;DBF=sample.db
```

The **Location** field can contain the equivalent of the SAP Sybase IQ Host connection parameter (for example, localhost:4444 or 10.25.99.253:2638).

```
Location: AppServer-pc:2639
```

The **Initial Catalog** field can contain the name of the database to connect to (for example, demo). The database must have been previously started.

```
Initial Catalog: demo
```

The combination of these last four fields and the user ID and password from the **Security** page must contain enough information to successfully connect to a database server.

3. Instead of specifying the database user ID and password as a connection parameter in the **Provider String** field where it would be exposed in plain text, you can fill in the **Security** page.

In SQL Server 2005/2008, click the **Be made using this security context** option and fill in the **Remote login** and **With password** fields (the password is displayed as asterisks).

4. Go to the **Server Options** page.

Enable the **RPC** and **RPC Out** options.

The technique for doing this varies with different versions of Microsoft SQL Server. In SQL Server 2000, there are two checkboxes that must be checked for these two options. In SQL Server 2005/2008, the options are True/False settings. Make sure that they are set True. The **Remote Procedure Call (RPC)** options must be set to execute stored procedure/function calls in an SAP Sybase IQ database and pass parameters in and out successfully.

5. Choose the **Allow Inprocess** provider option.

The technique for doing this varies with different versions of Microsoft SQL Server. In SQL Server 2000, there is a **Provider Options** button that takes you to the page where you can choose this option. For SQL Server 2005/2008, right-click the SAOLEDB.16 provider name under **Linked Servers » Providers** and click **Properties**. Make sure the **Allow Inprocess** checkbox is checked. If the **Inprocess** option is not chosen, queries fail.

6. Other provider options can be ignored. Several of these options pertain to SQL Server backwards compatibility and have no effect on the way SQL Server interacts with the SAP Sybase IQ OLE DB provider. Examples are **Nested queries** and **Supports LIKE operator**. Other options, when selected, may result in syntax errors or degraded performance.

The Microsoft Linked Server is configured.

Setting up a Linked Server Using a Script

A Linked Server definition may be set up using a SQL Server script.

Prerequisites

SQL Server 2005 or later.

Task

Make the appropriate changes to the following script using the steps below before running it on SQL Server.

```
USE [master]
GO
EXEC master.dbo.sp_adddlinkedserver @server=N'SADATABASE',
    @srvproduct=N'SAP Sybase IQ', @provider=N'SAOLEDB.16',
    @datasrc=datasrc=N'Sybase IQ Demo',
    @provstr=N'host=localhost:4444;server=myserver;dbn=demo'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
    @optname=N'rpc', @optvalue=N'true'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
    @optname=N'rpc out', @optvalue=N'true'
GO
-- Set remote login
EXEC master.dbo.sp_adddlinkedserverlogin @rmtsrvname = N'SADATABASE',
    @locallogin = NULL , @useself = N'False',
```

```

    @rmtuser = N'DBA', @rmtpassword = N'sql'
GO
-- Set global provider "allow in process" flag
EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.16',
N'AllowInProcess', 1

```

1. Choose a new Linked Server name (SADATABASE is used in the example).
2. Choose an optional data source name (SAP Sybase IQ 16 Demo is used in the example).
3. Choose an optional provider string (N'host=localhost:4444;server=myserver;dbn=demo' is used in the example).
4. Choose a remote user ID and password (N'DBA' and N'sql' are used in the example).

Your modified script can be run under Microsoft SQL Server to create a new Linked Server.

Supported OLE DB Interfaces

The OLE DB API consists of a set of interfaces. The following table describes the support for each interface in the SQL Anywhere OLE DB driver.

Interface	Purpose	Limitations
IAccessor	Define bindings between client memory and data store values.	DBACCESSOR_PASSBYREF not supported. DBACCESSOR_OPTIMIZED not supported.
IAlterIndex IAlterTable	Alter tables, indexes, and columns.	Not supported.
IChapteredRowset	A chaptered rowset allows rows of a rowset to be accessed in separate chapters.	Not supported. SAP Sybase IQ does not support chaptered rowsets.
IColumnsInfo	Get simple information about the columns in a rowset.	Supported.
IColumnsRowset	Get information about optional metadata columns in a rowset, and get a rowset of column metadata.	Supported.
ICommand	Execute SQL statements.	Does not support calling ICommandProperties: GetProperties with DBPROPSET_PROPERTIESINERROR to find properties that could not have been set.

Interface	Purpose	Limitations
ICommandPersist	Persist the state of a command object (but not any active rowsets). These persistent command objects can subsequently be enumerated using the PROCEDURES or VIEWS rowset.	Supported.
ICommandPrepare	Prepare commands.	Supported.
ICommandProperties	Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.	Supported.
ICommandText	Set the SQL statement text for ICommand.	Only the DBGUID_DEFAULT SQL dialect is supported.
ICommandWithParameters	Set or get parameter information for a command.	No support for parameters stored as vectors of scalar values. No support for BLOB parameters.
IConvertType		Supported.
IDBAsynchNotify IDBAsynchStatus	Asynchronous processing. Notify client of events in the asynchronous processing of data source initialization, populating rowsets, and so on.	Not supported.
IDBCreateCommand	Create commands from a session.	Supported.
IDBCreateSession	Create a session from a data source object.	Supported.
IDBDataSourceAdmin	Create/destroy/modify data source objects, which are COM objects used by clients. This interface is not used to manage data stores (databases).	Not supported.

Interface	Purpose	Limitations
IDBInfo	Find information about keywords unique to this provider (that is, to find non-standard SQL keywords). Also, find information about literals, special characters used in text matching queries, and other literal information.	Supported.
IDBInitialize	Initialize data source objects and enumerators.	Supported.
IDBProperties	Manage properties on a data source object or enumerator.	Supported.
IDBSchemaRowset	Get information about system tables, in a standard form (a rowset).	Supported.
IErrorInfo IErrorLookup IErrorRecords	ActiveX error object support.	Supported.
IGetDataSource	Returns an interface pointer to the session's data source object.	Supported.
IIndexDefinition	Create or drop indexes in the data store.	Not supported.
IMultipleResults	Retrieve multiple results (rowsets or row counts) from a command.	Supported.
IOpenRowset	Non-SQL way to access a database table by its name.	Supported. Opening a table by its name is supported, not by a GUID.
IParentRowset	Access chaptered/hierarchical rowsets.	Not supported.
IRowset	Access rowsets.	Supported.

Interface	Purpose	Limitations
IRowsetChange	Allow changes to rowset data, reflected back to the data store. InsertRow/SetData for BLOBs are not implemented.	Supported.
IRowsetChapterMember	Access chaptered/hierarchical rowsets.	Not supported.
IRowsetCurrentIndex	Dynamically change the index for a rowset.	Not supported.
IRowsetFind	Find a row within a rowset matching a specified value.	Not supported.
IRowsetIdentity	Compare row handles.	Not supported.
IRowsetIndex	Access database indexes.	Not supported.
IRowsetInfo	Find information about rowset properties or to find the object that created the rowset.	Supported.
IRowsetLocate	Position on rows of a rowset, using bookmarks.	Supported.
IRowsetNotify	Provides a COM callback interface for rowset events.	Supported.
IRowsetRefresh	Get the latest value of data that is visible to a transaction.	Not supported.
IRowsetResynch	Old OLE DB 1.x interface, superseded by IRowsetRefresh.	Not supported.
IRowsetScroll	Scroll through rowset to fetch row data.	Not supported.
IRowsetUpdate	Delay changes to rowset data until Update is called.	Supported.
IRowsetView	Use views on an existing rowset.	Not supported.
ISequentialStream	Retrieve a BLOB column.	Supported for reading only. No support for SetData with this interface.

Interface	Purpose	Limitations
ISessionProperties	Get session property information.	Supported.
ISourcesRowset	Get a rowset of data source objects and enumerators.	Supported.
ISQLErrorInfo ISupportErrorInfo	ActiveX error object support.	Supported.
ITableDefinition ITableDefinitionWithConstraints	Create, drop, and alter tables, with constraints.	Supported.
ITransaction	Commit or abort transactions.	Not all the flags are supported.
ITransactionJoin	Support distributed transactions.	Not all the flags are supported.
ITransactionLocal	Handle transactions on a session. Not all the flags are supported.	Supported.
ITransactionOptions	Get or set options on a transaction.	Supported.
IViewChapter	Work with views on an existing rowset, specifically to apply post-processing filters/sorting on rows.	Not supported.
IViewFilter	Restrict contents of a rowset to rows matching a set of conditions.	Not supported.
IViewRowset	Restrict contents of a rowset to rows matching a set of conditions, when opening a rowset.	Not supported.
IViewSort	Apply sort order to a view.	Not supported.

OLE DB Provider Registration

When the SAOLEDB provider is installed using the SAP Sybase IQ installer, the provider registers itself. This registration process includes making registry entries in the COM section

of the registry, so that ADO can locate the DLL when the SAOLEDB provider is called. If you change the location of your DLL, you must re-register it.

Example

The following commands register the SAP Sybase IQ OLE DB provider when run from the directory where the provider is installed:

```
regsvr32 dboledb16.dll  
regsvr32 dboledba16.dll
```

ODBC CLI

ODBC (Open Database Connectivity) is a standard call level interface (CLI) developed by Microsoft Corporation. It is based on the SQL Access Group CLI specification. ODBC applications can run against any data source that provides an ODBC driver. ODBC is a good choice for a programming interface if you want your application to be portable to other data sources that have ODBC drivers.

ODBC conformance

SAP Sybase IQ provides support for ODBC 3.5, which is supplied as part of the Microsoft Data Access Kit 2.7.

Levels of ODBC support

ODBC features are arranged according to level of conformance. Features are either **Core**, **Level 1**, or **Level 2**, with Level 2 being the most complete level of ODBC support. These features are listed in the Microsoft *ODBC Programmer's Reference* at <http://msdn.microsoft.com/en-us/library/ms714177.aspx>.

Features supported by SAP Sybase IQ

SAP Sybase IQ supports the ODBC 3.5 specification as follows:

- **Core conformance** – SAP Sybase IQ supports all Core level features.
- **Level 1 conformance** – SAP Sybase IQ supports all Level 1 features, except for asynchronous execution of ODBC functions.

SAP Sybase IQ supports multiple threads sharing a single connection. The requests from the different threads are serialized by SAP Sybase IQ.

- **Level 2 conformance** – SAP Sybase IQ supports all Level 2 features, except for the following ones:
 - Three part names of tables and views. This is not applicable for SAP Sybase IQ.
 - Asynchronous execution of ODBC functions for specified individual statements.
 - Ability to time out login requests and SQL queries.

ODBC application development

Every C/C++ source file that calls ODBC functions must include a platform-specific ODBC header file. Each platform-specific header file includes the main ODBC header file `odbc.h`,

which defines all the functions, data types, and constant definitions required to write an ODBC program.

Perform the following tasks to include the ODBC header file in a C/C++ source file:

1. Add an include line referencing the appropriate platform-specific header file to your source file. The lines to use are as follows:

Operating system	Include line
Windows	#include "ntodbc.h"
Unix	#include "unixodbc.h"
Windows Mobile	#include "ntodbc.h"

2. Add the directory containing the header file to the include path for your compiler. Both the platform-specific header files and `odbc.h` are installed in the `SDK\Include` subdirectory of your SAP Sybase IQ installation directory.
3. When building ODBC applications for Unix, you might have to define the macro "UNIX" for 32-bit applications or "UNIX64" for 64-bit applications to obtain the correct data alignment and sizes. This step is not required if you are using one of the following supported compilers:

- GNU C/C++ compiler on any supported platform
- Intel C/C++ compiler for Linux (`icc`)
- SunPro C/C++ compiler for Linux or Solaris
- VisualAge C/C++ compiler for AIX
- C/C++ compiler (`cc/aCC`) for HP-UX

Once your source code has been written, you are ready to compile and link the application. The following sections describe how to create executable applications.

ODBC Applications on Windows

When linking your application, you must link against the appropriate import library file to have access to the ODBC functions.

The import library defines entry points for the ODBC driver manager `odbc32.dll`. The driver manager in turn loads the SAP Sybase IQ ODBC driver `dbodbc16.dll`.

Typically, the import library is stored under the `Lib` directory structure of the Microsoft platform SDK:

Operating system	Import library
Windows (32-bit)	<code>\Lib\X86\odbc32.lib</code>
Windows (64-bit)	<code>\Lib\X86\odbc32.lib</code>

Example

The following command illustrates how to add the directory containing the platform-specific import library to the list of library directories in your LIB environment variable:

```
set LIB=%LIB%;c:\mssdk\v7.0\lib
```

The following command illustrates how to compile and link the application stored in `odbc.c` using the Microsoft compile and link tool:

```
cl odbc.c /I"%IQDIR16%\SDK\Lib\X86\Include" odbc32.lib
```

ODBC applications on Unix

An ODBC driver manager for Unix is included with SAP Sybase IQ and there are third party driver managers available. This section describes how to build ODBC applications that do not use an ODBC driver manager.

ODBC driver

The ODBC driver is a shared object or shared library. Separate versions of the SAP Sybase IQ ODBC driver are supplied for single-threaded and multithreaded applications. A generic SAP Sybase IQ ODBC driver is supplied that will detect the threading model in use and direct calls to the appropriate single-threaded or multithreaded library.

The ODBC drivers are the following files:

Operating system	Threading model	ODBC driver
(all Unix except HP-UX)	Generic	libdbodbc16.so (libdbodbc16.so.1)
(all Unix except HP-UX)	Single threaded	libdbodbc16_n.so (libdbodbc16_n.so.1)
(all Unix except HP-UX)	Multithreaded	libdbodbc16_r.so (libdbodbc16_r.so.1)
HP-UX	Generic	libdbodbc16.sl (libdbodbc16.sl.1)
HP-UX	Single threaded	libdbodbc16_n.sl (libdbodbc16_n.sl.1)
HP-UX	Multithreaded	libdbodbc16_r.sl (libdbodbc16_r.sl.1)

The libraries are installed as symbolic links to the shared library with a version number (shown in parentheses).

When linking an ODBC application on Unix, link your application against the generic ODBC driver `libdbodbc16`. When deploying your application, ensure that the appropriate (or all) ODBC driver versions (non-threaded or threaded) are available in the user's library path.

Data source information

If SAP Sybase IQ does not detect the presence of an ODBC driver manager, it uses the system information file for data source information.

The unixODBC driver manager

Versions of the unixODBC release before version 2.2.14 have incorrectly implemented some aspects of the 64-bit ODBC specification as defined by Microsoft. These differences will cause problems when using the unixODBC driver manager with the SAP Sybase IQ 64-bit ODBC driver.

To avoid these problems, you should be aware of the differences. One of them is the definition of `SQLLEN` and `SQLULEN`. These are 64-bit types in the Microsoft 64-bit ODBC specification, and are expected to be 64-bit quantities by the SAP Sybase IQ 64-bit ODBC driver. Some implementations of unixODBC define these two types as 32-bit quantities and this will result in problems when interfacing to the SAP Sybase IQ 64-bit ODBC driver.

There are three things that you must do to avoid problems on 64-bit platforms.

1. Instead of including the unixODBC headers like `sql.h` and `sqlext.h`, you should include the SAP Sybase IQ ODBC header file `unixodbc.h`. This will guarantee that you have the correct definitions for `SQLLEN` and `SQLULEN`. The header files in unixODBC 2.2.14 or later versions correct this problem.
2. You must ensure that you have used the correct types for all parameters. Use of the correct header file and the strong type checking of your C/C++ compiler should help in this area. You must also ensure that you have used the correct types for all variables that are set by the SAP Sybase IQ driver indirectly through pointers.
3. Do not use versions of the unixODBC driver manager before release 2.2.14. Link directly to the SAP Sybase IQ ODBC driver instead. For example, ensure that the `libodbc` shared object is linked to the SAP Sybase IQ driver.

```
libodbc.so.1 -> libdbodbc16_r.so.1
```

Alternatively, you can use the SAP Sybase IQ driver manager on platforms where it is available.

UTF-32 ODBC driver managers for Unix

Versions of ODBC driver managers that define `SQLWCHAR` as 32-bit (UTF-32) quantities cannot be used with the SAP Sybase IQ ODBC driver that supports wide calls since this driver is built for 16-bit `SQLWCHAR`. For these cases, an ANSI-only version of the SAP Sybase IQ

ODBC driver is provided. This version of the ODBC driver does not support the wide call interface (for example, SQLConnectW).

The shared object name of the driver is `libdbodbcansi16_r`. Only a threaded variant of the driver is provided. Certain frameworks, such as Real Basic, do not work with the dylib and require the bundle.

The regular ODBC driver treats SQLWCHAR strings as UTF-16 strings. This driver cannot be used with some ODBC driver managers, such as iODBC, which treat SQLWCHAR strings as UTF-32 strings. When dealing with Unicode-enabled drivers, these driver managers translate narrow calls from the application to wide calls into the driver. An ANSI-only driver gets around this behavior, allowing the driver to be used with such driver managers, as long as the application does not make any wide calls. Wide calls through iODBC, or any other driver manager with similar semantics, remain unsupported.

ODBC Samples

Several ODBC samples are included with SAP Sybase IQ. You can find the samples in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C` directory (Windows) and `$SYBASE/IQ-16_0/samples/sqlanywhere/c` directory (UNIX).

The samples in directories starting with ODBC illustrate separate and simple ODBC tasks, such as connecting to a database and executing statements. A complete sample ODBC program is supplied in the `odbc.c` file. This program performs the same actions as the embedded SQL dynamic cursor example program that is in the same directory.

Building the Sample ODBC Program for Windows

Building the sample ODBC program allows you to run the program and see how it performs ODBC tasks, such as connecting to a database and executing statements.

Prerequisites

For x64 platform builds, you may need to set up the correct environment for compiling and linking. Here is an example that builds the sample programs for an x64 platform.

```
set mssdk=c:\mssdk\v7.0
build64
```

Task

A batch file located in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C` directory can be used to compile and link all the sample applications.

1. Open a command prompt and change the directory to the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C` directory.

ODBC CLI

2. Run the `build.bat` or `build64.bat` batch file.

The sample ODBC program is built.

Building the Sample ODBC Program for Unix

Building the sample ODBC program allows you to run the program and see how it performs ODBC tasks, such as connecting to a database and executing statements.

Prerequisites

There are no prerequisites for this task.

Task

A shell script located in the `$SYBASE/IQ-16_0/samples/sqlanywhere/c` directory can be used to compile and link all the sample applications.

1. Open a command shell and change the directory to the `$SYBASE/IQ-16_0/samples/sqlanywhere/c` directory.
2. Run the `build.sh` shell script.

The sample ODBC program is built.

ODBC Sample Programs

You can load the sample ODBC program by running the file on the appropriate platform.

- For 32-bit Windows, run `%ALLUSERSPROFILE%\SybaseIQ\samples\sqlanywhere\C\odbcwin.exe`.
- For 64-bit Windows, run `%ALLUSERSPROFILE%\SybaseIQ\samples\sqlanywhere\C\odbcx64.exe`.
- For Unix, run `$SYBASE/IQ-16_0/samples/sqlanywhere/C/odbc`.

After running the file, choose one of the tables in the sample database. For example, you can enter `Customers` or `Employees`.

ODBC handles

ODBC applications use a small set of **handles** to define basic features such as database connections and SQL statements. A handle is a 32-bit value.

The following handles are used in essentially all ODBC applications:

- **Environment** – The environment handle provides a global context in which to access data. Every ODBC application must allocate exactly one environment handle upon starting, and must free it at the end.

The following code illustrates how to allocate an environment handle:

```
SQLRETURN rc;
SQLHENV env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

- **Connection** – A connection is specified by an ODBC driver and a data source. An application can have several connections associated with its environment. Allocating a connection handle does not establish a connection; a connection handle must be allocated first and then used when the connection is established.

The following code illustrates how to allocate a connection handle:

```
SQLRETURN rc;
SQLHDBC dbc;
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

- **Statement** – A statement handle provides access to a SQL statement and any information associated with it, such as result sets and parameters. Each connection can have several statements. Statements are used both for cursor operations (fetching data) and for single statement execution (for example, INSERT, UPDATE, and DELETE).

The following code illustrates how to allocate a statement handle:

```
SQLRETURN rc;
SQLHSTMT stmt;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

How to allocate ODBC handles

The handle types required for ODBC programs are as follows:

Item	Handle type
Environment	SQLHENV
Connection	SQLHDBC
Statement	SQLHSTMT
Descriptor	SQLHDESC

To use an ODBC handle, you perform the following tasks:

1. Call the `SQLAllocHandle` function.
2. Use the handle in subsequent function calls.
3. Free the object using `SQLFreeHandle`.

`SQLAllocHandle` takes the following parameters:

- an identifier for the type of item being allocated
- the handle of the parent item
- a pointer to the location of the handle to be allocated

ODBC CLI

For information, see `SQLAllocHandle` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms712455.aspx>.

`SQLFreeHandle` takes the following parameters:

- an identifier for the type of item being freed
- the handle of the item being freed

For information, see `SQLFreeHandle` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms710123.aspx>.

Example

The following code fragment allocates and frees an environment handle:

```
SQLRETURN rc;
SQLHENV env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
if( rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO )
{
    .
    .
    .
}
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

ODBC example

A simple ODBC program that connects to the SAP Sybase IQ sample database and immediately disconnects can be found in `%IQDIRSAMP16%\SQLAnywhere\ODBCConnect\odbcconnect.cpp`. This example shows the steps required in setting up the environment to make a connection to a database server, as well as the steps required in disconnecting from the server and freeing up resources.

ODBC Connection Functions

ODBC supplies a set of connection functions. Which one you use depends on how you expect your application to be deployed and used:

- **SQLConnect** – The simplest connection function.

`SQLConnect` takes a data source name and optional user ID and password. You may want to use `SQLConnect` if you hard-code a data source name into your application.

For more information, see `SQLConnect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms711810.aspx>.

- **SQLDriverConnect** – Connects to a data source using a connection string.

`SQLDriverConnect` allows the application to use SAP Sybase IQ-specific connection information that is external to the data source. Also, you can use `SQLDriverConnect` to request that the Sybase IQ ODBC driver driver prompt for connection information.

SQLDriverConnect can also be used to connect without specifying a data source. The Sybase IQ ODBC driver name is specified instead. The following example connects to a server and database that is already running.

```
SQLSMALLINT cso;
SQLCHAR     scso[2048];

SQLDriverConnect( hdbc, NULL,
    "Driver=Sybase IQ;UID=<user_id>;PWD=<password>", SQL_NTS,
    scso, sizeof(scso)-1,
    &cso, SQL_DRIVER_NOPROMPT );
```

For more information, see `SQLDriverConnect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms715433.aspx>.

- **SQLBrowseConnect** – Connects to a data source using a connection string, like `SQLDriverConnect`.

`SQLBrowseConnect` allows your application to build its own windows to prompt for connection information and to browse for data sources used by a particular driver (in this case the Sybase IQ ODBC driver).

For more information, see `SQLBrowseConnect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms714565.aspx>.

Establishing an ODBC Connection

Establish an ODBC connection in your application to perform any database operations.

Prerequisites

There are no prerequisites for this task.

Task

You can find a complete sample in `%ALLUSERSPROFILE%\SybaseIQ\samples\ODBCConnect\odbconnect.cpp`.

1. Allocate an ODBC environment.

For example:

```
SQLRETURN rc;
SQLHENV   env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

2. Declare the ODBC version.

By declaring that the application follows ODBC version 3, `SQLSTATE` values and some other version-dependent features are set to the proper behavior. For example:

```
rc = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION,
    (void*)SQL_OV_ODBC3, 0 );
```

3. Allocate an ODBC connection item.

For example:

```
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

4. Set any connection attributes that must be set before connecting.

Some connection attributes must be set before establishing a connection or after establishing a connection, while others can be set either before or after. The `SQL_AUTOCOMMIT` attribute is one that can be set before or after:

```
rc = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
```

By default, ODBC operates in autocommit mode. This mode is turned off by setting `SQL_AUTOCOMMIT` to false.

5. If necessary, assemble the data source or connection string.

Depending on your application, you may have a hard-coded data source or connection string, or you may store it externally for greater flexibility.

6. Call the ODBC connection function.

For example:

```
if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
{
    printf( "dbc allocated\n" );
    rc = SQLConnect( dbc,
        (SQLCHAR *) "Sybase IQ Demo", SQL_NTS,
        (SQLCHAR *) "<user_id>", SQL_NTS,
        (SQLCHAR *) "<password>", SQL_NTS );
    if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
    {
        // Successfully connected.
    }
}
```

Every string passed to ODBC has a corresponding length. If the length is unknown, you can pass `SQL_NTS` indicating that it is a *Null Terminated String* whose end is marked by the null character (`\0`).

The application, when built and run, establishes an ODBC connection.

Server options changed by ODBC

The SAP Sybase IQ ODBC driver sets some temporary server options when connecting to an SAP Sybase IQ database. The following options are set as indicated.

- **date_format** – yyyy-mm-dd
- **date_order** – ymd
- **isolation_level** – based on the `SQL_ATTR_TXN_ISOLATION/SA_SQL_ATTR_TXN_ISOLATION` attribute setting of `SQLSetConnectAttr`. The following options are available.

```
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_READ_COMMITTED
```

```
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE
SA_SQL_TXN_SNAPSHOT
SA_SQL_TXN_STATEMENT_SNAPSHOT
SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
```

- **time_format** – hh:nn:ss
- **timestamp_format** – yyyy-mm-dd hh:nn:ss.ssssss
- **timestamp_with_time_zone_format** – yyyy-mm-dd hh:nn:ss.ssssss +hh:nn

To restore the default option setting, execute a SET statement. Here is an example of a statement that will reset the `timestamp_format` option.

```
set temporary option timestamp_format =
```

SQLSetConnectAttr extended connection attributes

The SAP Sybase IQ ODBC driver supports some extended connection attributes.

- **SA_REGISTER_MESSAGE_CALLBACK** – Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

A message handler routine can be created to intercept these messages. The message handler callback prototype is as follows:

```
void SQL_CALLBACK message_handler(
SQLHDBC sqlany_dbc,
unsigned char msg_type,
long code,
unsigned short length,
char * message
);
```

The following possible values for `msg_type` are defined in `sqldef.h`.

- **MESSAGE_TYPE_INFO** – The message type was INFO.
- **MESSAGE_TYPE_WARNING** – The message type was WARNING.
- **MESSAGE_TYPE_ACTION** – The message type was ACTION.
- **MESSAGE_TYPE_STATUS** – The message type was STATUS.
- **MESSAGE_TYPE_PROGRESS** – The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

A SQLCODE associated with the message may be provided in `code`. When not available, the `code` parameter value is 0.

The length of the message is contained in `length`.

A pointer to the message is contained in `message`. The message string is not null-terminated. Your application must be designed to handle this. The following is an example.

```
memcpy( mybuff, msg, len );
mybuff[ len ] = '\0';
```

To register the message handler in ODBC, call the `SQLSetConnectAttr` function as follows:

```
rc = SQLSetConnectAttr(
    hdbc,
    SA_REGISTER_MESSAGE_CALLBACK,
    (SQLPOINTER) &message_handler, SQL_IS_POINTER );
```

To unregister the message handler in ODBC, call the `SQLSetConnectAttr` function as follows:

```
rc = SQLSetConnectAttr(
    hdbc,
    SA_REGISTER_MESSAGE_CALLBACK,
    NULL, SQL_IS_POINTER );
```

- **SA_GET_MESSAGE_CALLBACK_PARM** – To retrieve the value of the SQLHDBC connection handle that will be passed to message handler callback routine, use `SQLGetConnectAttr` with the `SA_GET_MESSAGE_CALLBACK_PARM` parameter.

```
SQLHDBC callback_hdbc = NULL;
rc = SQLGetConnectAttr(
    hdbc,
    SA_GET_MESSAGE_CALLBACK_PARM,
    (SQLPOINTER) &callback_hdbc, 0, 0 );
```

The returned value will be the same as the parameter value that is passed to the message handler callback routine.

- **SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK** – This is used to register a file transfer validation callback function. Before allowing any transfer to take place, the ODBC driver will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the ODBC driver will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below.

The callback prototype is as follows:

```
int SQL_CALLBACK file_transfer_callback(
    void * sqlca,
    char * file_name,
    int is_write
);
```

The `file_name` parameter is the name of the file to be read or written. The `is_write` parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When

initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the ODBC driver allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. If the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

- **SA_SQL_ATTR_TXN_ISOLATION** – This is used to set an extended transaction isolation level. The following example sets a Snapshot isolation level:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SA_SQL_ATTR_TXN_ISOLATION,
    SA_SQL_TXN_SNAPSHOT, SQL_IS_INTEGER );
```

64-bit ODBC considerations

When you use an ODBC function like `SQLBindCol`, `SQLBindParameter`, or `SQLGetData`, some of the parameters are typed as `SQLLEN` or `SQLULEN` in the function prototype. Depending on the date of the Microsoft ODBC API Reference documentation that you are looking at, you might see the same parameters described as `SQLINTEGER` or `SQLUINTEGER`.

`SQLLEN` and `SQLULEN` data items are 64 bits in a 64-bit ODBC application and 32 bits in a 32-bit ODBC application. `SQLINTEGER` and `SQLUINTEGER` data items are 32 bits on all platforms.

To illustrate the problem, the following ODBC function prototype was excerpted from an older copy of the Microsoft ODBC API Reference.

```
SQLRETURN SQLGetData (
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT  ColumnNumber,
    SQLSMALLINT   TargetType,
    SQLPOINTER    TargetValuePtr,
    SQLINTEGER    BufferLength,
    SQLINTEGER    *StrLen_or_IndPtr);
```

Compare this with the actual function prototype found in `sql.h` in Microsoft Visual Studio version 8.

```
SQLRETURN SQL_API SQLGetData (
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT  ColumnNumber,
    SQLSMALLINT   TargetType,
    SQLPOINTER    TargetValue,
```

```

SQLLEN      BufferLength,
SQLLEN      *StrLen_or_Ind);

```

As you can see, the `BufferLength` and `StrLen_or_Ind` parameters are now typed as `SQLLEN`, not `SQLINTEGER`. For the 64-bit platform, these are 64-bit quantities, not 32-bit quantities as indicated in the Microsoft documentation.

To avoid issues with cross-platform compilation, SAP Sybase IQ provides its own ODBC header files. For Windows platforms, you should include the `ntodbc.h` header file. For Unix platforms such as Linux, you should include the `unixodbc.h` header file. Use of these header files ensures compatibility with the corresponding SAP Sybase IQ ODBC driver for the target platform.

The following table lists some common ODBC types that have the same or different storage sizes on 64-bit and 32-bit platforms.

ODBC API	64-bit platform	32-bit platform
SQLINTEGER	32 bits	32 bits
SQLUINTEGER	32 bits	32 bits
SQLLEN	64 bits	32 bits
SQLULEN	64 bits	32 bits
SQLSETPOSIROW	64 bits	16 bits
SQL_C_BOOKMARK	64 bits	32 bits
BOOKMARK	64 bits	32 bits

If you declare data variables and parameters incorrectly, then you may encounter incorrect software behavior.

The following table summarizes the ODBC API function prototypes that have changed with the introduction of 64-bit support. The parameters that are affected are noted. The parameter name as documented by Microsoft is shown in parentheses when it differs from the actual parameter name used in the function prototype. The parameter names are those used in the Microsoft Visual Studio version 8 header files.

ODBC API	Parameter (documented parameter name)
SQLBindCol	SQLLEN BufferLength SQLLEN *Strlen_or_Ind
SQLBindParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind

ODBC API	Parameter (documented parameter name)
SQLBindParameter	SQLULEN cbColDef (ColumnSize) SQLLEN cbValueMax (BufferLength) SQLLEN *pcbValue (Strlen_or_IndPtr)
SQLColAttribute	SQLLEN *NumericAttribute
SQLColAttributes	SQLLEN *pfDesc
SQLDescribeCol	SQLULEN *ColumnSize (ColumnSizePtr)
SQLDescribeParam	SQLULEN *pcbParamDef (ParameterSizePtr)
SQLExtendedFetch	SQLLEN irow (FetchOffset) SQLULEN *pcrow (RowCountPtr)
SQLFetchScroll	SQLLEN FetchOffset
SQLGetData	SQLLEN BufferLength SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLGetDescRec	SQLLEN *Length (LengthPtr)
SQLParamOptions	SQLULEN crow, SQLULEN *pirow
SQLPutData	SQLLEN Strlen_or_Ind
SQLRowCount	SQLLEN *RowCount (RowCountPtr)
SQLSetConnectOption	SQLULEN Value
SQLSetDescRec	SQLLEN Length SQLLEN *StringLength (StringLengthPtr) SQLLEN *Indicator (IndicatorPtr)
SQLSetParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLSetPos	SQLSETPOSIROW irow (RowNumber)
SQLSetScrollOptions	SQLLEN crowKeyset
SQLSetStmtOption	SQLULEN Value

ODBC CLI

Some values passed into and returned from ODBC API calls through pointers have changed to accommodate 64-bit applications. For example, the following values for the SQLSetStmtAttr and SQLSetDescField functions are no longer SQLINTEGER/SQLINTEGER. The same rule applies to the corresponding parameters for the SQLGetStmtAttr and SQLGetDescField functions.

ODBC API	Type for Value/ValuePtr variable
SQLSetStmtAttr(SQL_ATTR_FETCH_BOOK-MARK_PTR)	SQLLEN * value
SQLSetStmtAttr(SQL_ATTR_KEYSET_SIZE)	SQLULEN value
SQLSetStm-tAttr(SQL_ATTR_MAX_LENGTH)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_ROWS)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_PAR-AM_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMS_PRO-CESSED_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARA-MSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_AR-RAY_SIZE)	SQLULEN value
SQLSetStm-tAttr(SQL_ATTR_ROW_BIND_OFF-SET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_ROW_NUM-BER)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROWS_FETCH-ED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_ARRAY_SIZE)	SQLULEN value
SQLSetDescField(SQL_DESC_BIND_OFF-SET_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_ROWS_PRO-CESSED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_DIS-PLAY_SIZE)	SQLLEN value

ODBC API	Type for Value/ValuePtr variable
SQLSetDescField(SQL_DESC_INDICATOR_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH_PTR)	SQLLEN * value

For more information, see the Microsoft article "ODBC 64-Bit API Changes in MDAC 2.7" at <http://support.microsoft.com/kb/298678>.

Data alignment requirements

When you use SQLBindCol, SQLBindParameter, or SQLGetData, a C data type is specified for the column or parameter. On certain platforms, the storage (memory) provided for each column must be properly aligned to fetch or store a value of the specified type. The ODBC driver checks for proper data alignment. When an object is not properly aligned, the ODBC driver will issue an **"Invalid string or buffer length"** message (SQLSTATE HY090 or S1090).

The following table lists memory alignment requirements for processors such as Sun Sparc, Itanium-IA64, and ARM-based devices. The memory address of the data value must be a multiple of the indicated value.

C data type	Alignment required
SQL_C_CHAR	none
SQL_C_BINARY	none
SQL_C_GUID	none
SQL_C_BIT	none
SQL_C_STINYINT	none
SQL_C_UTINYINT	none
SQL_C_TINYINT	none
SQL_C_NUMERIC	none
SQL_C_DEFAULT	none
SQL_C_SSHORT	2

C data type	Alignment required
SQL_C_USHORT	2
SQL_C_SHORT	2
SQL_C_DATE	2
SQL_C_TIME	2
SQL_C_TIMESTAMP	2
SQL_C_TYPE_DATE	2
SQL_C_TYPE_TIME	2
SQL_C_TYPE_TIMESTAMP	2
SQL_C_WCHAR	2 (buffer size must be a multiple of 2 on all platforms)
SQL_C_SLONG	4
SQL_C_ULONG	4
SQL_C_LONG	4
SQL_C_FLOAT	4
SQL_C_DOUBLE	8 (4 for ARM)
SQL_C_SBIGINT	8
SQL_C_UBIGINT	8

The x86, x64, and PowerPC platforms do not require memory alignment. The x64 platform includes Advanced Micro Devices (AMD) AMD64 processors and Intel Extended Memory 64 Technology (EM64T) processors.

Result sets in ODBC applications

ODBC applications use cursors to manipulate and update result sets. SAP Sybase IQ provides extensive support for different kinds of cursors and cursor operations.

ODBC transaction isolation levels

You can use `SQLSetConnectAttr` to set the transaction isolation level for a connection. The characteristics that determine the transaction isolation level that SAP Sybase IQ provides include the following:

- **SQL_TXN_READ_UNCOMMITTED** – Set isolation level to 0. When this attribute value is set, it isolates any data read from changes by others and changes made by others

cannot be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read. This is the default value for isolation level.

- **SQL_TXN_READ_COMMITTED** – Set isolation level to 1. When this attribute value is set, it does not isolate data read from changes by others, and changes made by others can be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read.
- **SQL_TXN_REPEATABLE_READ** – Set isolation level to 2. When this attribute value is set, it isolates any data read from changes by others, and changes made by others cannot be seen. The re-execution of the read statement is affected by others. This supports a repeatable read.
- **SQL_TXN_SERIALIZABLE** – Set isolation level to 3. When this attribute value is set, it isolates any data read from changes by others, and changes made by others cannot be seen. The re-execution of the read statement is not affected by others. This supports a repeatable read.
- **SA_SQL_TXN_SNAPSHOT** – Set isolation level to Snapshot. When this attribute value is set, it provides a single view of the database for the entire transaction.
- **SA_SQL_TXN_STATEMENT_SNAPSHOT** – Set isolation level to Statement-snapshot. When this attribute value is set, it provides less consistency than Snapshot isolation, but may be useful when long running transactions result in too much space being used in the temporary file by the version store.
- **SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT** – Set isolation level to Readonly-statement-snapshot. When this attribute value is set, it provides less consistency than Statement-snapshot isolation, but avoids the possibility of update conflicts. Therefore, it is most appropriate for porting applications originally intended to run under different isolation levels.

The `allow_snapshot_isolation` database option must be set to On to use the Snapshot, Statement-snapshot, or Readonly-statement-snapshot settings.

For more information, see `SQLSetConnectAttr` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms713605.aspx>.

Example

The following fragment sets the isolation level to Snapshot:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION,
    SA_SQL_TXN_SNAPSHOT, SQL_IS_INTEGER );
```

ODBC cursor characteristics

ODBC functions that execute statements and manipulate result sets, use cursors to perform their tasks. Applications open a cursor implicitly whenever they execute a `SQLExecute` or `SQLExecDirect` function.

For applications that move through a result set only in a forward direction and do not update the result set, cursor behavior is relatively straightforward. By default, ODBC applications

request this behavior. ODBC defines a read-only, forward-only cursor, and SAP Sybase IQ provides a cursor optimized for performance in this case.

For applications that need to scroll both forward and backward through a result set, such as many graphical user interface applications, cursor behavior is more complex. What does the application when it returns to a row that has been updated by some other application? ODBC defines a variety of **scrollable cursors** to allow you to build in the behavior that suits your application. SAP Sybase IQ provides a full set of cursors to match the ODBC scrollable cursor types.

You set the required ODBC cursor characteristics by calling the `SQLSetStmtAttr` function that defines statement attributes. You must call `SQLSetStmtAttr` before executing a statement that creates a result set.

You can use `SQLSetStmtAttr` to set many cursor characteristics. The characteristics that determine the cursor type that SAP Sybase IQ supplies include the following:

- **SQL_ATTR_CURSOR_SCROLLABLE** – Set to `SQL_SCROLLABLE` for a scrollable cursor and `SQL_NONSCROLLABLE` for a forward-only cursor. `SQL_NONSCROLLABLE` is the default.
- **SQL_ATTR_CONCURRENCY** – Set to one of the following values:
 - **SQL_CONCUR_READ_ONLY** – Disallow updates. `SQL_CONCUR_READ_ONLY` is the default.
 - **SQL_CONCUR_LOCK** – Use the lowest level of locking sufficient to ensure that the row can be updated.
 - **SQL_CONCUR_ROWVER** – Use optimistic concurrency control, comparing row versions such as SQLBase ROWID or Sybase TIMESTAMP.
 - **SQL_CONCUR_VALUES** – Use optimistic concurrency control, comparing values.

For more information, see `SQLSetStmtAttr` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms712631.aspx>.

Example

The following fragment requests a read-only, scrollable cursor:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE,
                SQL_SCROLLABLE, SQL_IS_INTEGER );
```

Data retrieval

To retrieve rows from a database, you execute a `SELECT` statement using `SQLExecute` or `SQLExecDirect`. This opens a cursor on the statement.

You then use `SQLFetch` or `SQLFetchScroll` to fetch rows through the cursor. These functions fetch the next rowset of data from the result set and return data for all bound columns. Using `SQLFetchScroll`, rowsets can be specified at an absolute or relative position or by bookmark. `SQLFetchScroll` replaces the older `SQLExtendedFetch` from the ODBC 2.0 specification.

When an application frees the statement using `SQLFreeHandle`, it closes the cursor.

To fetch values from a cursor, your application can use either `SQLBindCol` or `SQLGetData`. If you use `SQLBindCol`, values are automatically retrieved on each fetch. If you use `SQLGetData`, you must call it for each column after each fetch.

`SQLGetData` is used to fetch values in pieces for columns such as `LONG VARCHAR` or `LONG BINARY`. As an alternative, you can set the `SQL_ATTR_MAX_LENGTH` statement attribute to a value large enough to hold the entire value for the column. The default value for `SQL_ATTR_MAX_LENGTH` is 256 KB.

The SAP Sybase IQ ODBC driver implements `SQL_ATTR_MAX_LENGTH` in a different way than intended by the ODBC specification. The intended meaning for `SQL_ATTR_MAX_LENGTH` is that it be used as a mechanism to truncate large fetches. This might be done for a "preview" mode where only the first part of the data is displayed. For example, instead of transmitting a 4 MB blob from the server to the client application, only the first 500 bytes of it might be transmitted (by setting `SQL_ATTR_MAX_LENGTH` to 500). The SAP Sybase IQ ODBC driver does not support this implementation.

The following code fragment opens a cursor on a query and retrieves data through the cursor. Error checking has been omitted to make the example easier to read. The fragment is taken from a complete sample, which can be found in `%IQDIRSAMP16%\SQLAnywhere\ODBCSelect\odbcselect.cpp`.

```
SQLINTEGER cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];
SQLSMALLINT deptID, managerID;
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN rc;

SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
SQLSetEnvAttr( env,
               SQL_ATTR_ODBC_VERSION,
               (void *)SQL_OV_ODBC3, 0 );
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLConnect( dbc,
            (SQLCHAR *) "SAP Sybase IQ 16 Demo", SQL_NTS,
            (SQLCHAR *) "DBA", SQL_NTS,
            (SQLCHAR *) "sql", SQL_NTS );
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLBindCol( stmt, 1,
            SQL_C_SSHORT, &deptID, 0, &cbDeptID );
SQLBindCol( stmt, 2,
            SQL_C_CHAR, deptName,
            sizeof(deptName), &cbDeptName );
SQLBindCol( stmt, 3,
            SQL_C_SSHORT, &managerID, 0, &cbManagerID );
SQLExecDirect( stmt, (SQLCHAR * )
              "SELECT DepartmentID, DepartmentName, DepartmentHeadID "
              "FROM Departments "
              "ORDER BY DepartmentID", SQL_NTS );
```

```

while( ( rc = SQLFetch( stmt ) ) != SQL_NO_DATA )
{
    printf( "%d %20s %d\n", deptID, deptName, managerID );
}
SQLFreeHandle( SQL_HANDLE_STMT, stmt );
SQLDisconnect( dbc );
SQLFreeHandle( SQL_HANDLE_DBC, dbc );
SQLFreeHandle( SQL_HANDLE_ENV, env );

```

The number of row positions you can fetch in a cursor is governed by the size of an integer. You can fetch rows numbered up to number 2147483646, which is one less than the value that can be held in a 32-bit integer. When using negative numbers (rows from the end) you can fetch down to one more than the largest negative value that can be held in an integer.

Row updates and deletes through a cursor

The Microsoft *ODBC Programmer's Reference* suggests that you use SELECT...FOR UPDATE to indicate that a query is updatable using positioned operations. You do not need to use the FOR UPDATE clause in SAP Sybase IQ: SELECT statements are automatically updatable as long as the following conditions are met:

- The underlying query supports updates.
That is to say, as long as a data manipulation statement on the columns in the result is meaningful, then positioned data manipulation statements can be carried out on the cursor. The `ansi_update_constraints` database option limits the type of queries that are updatable.
- The cursor type supports updates.
If you are using a read-only cursor, you cannot update the result set.

ODBC provides two alternatives for carrying out positioned updates and deletes:

- Use the `SQLSetPos` function.
Depending on the parameters supplied (`SQL_POSITION`, `SQL_REFRESH`, `SQL_UPDATE`, `SQL_DELETE`) `SQLSetPos` sets the cursor position and allows an application to refresh data, or update, or delete data in the result set. This is the method to use with SAP Sybase IQ.
- Send positioned UPDATE and DELETE statements using `SQLExecute`. This method should not be used with SAP Sybase IQ.

Bookmarks

ODBC provides **bookmarks**, which are values used to identify rows in a cursor. SAP Sybase IQ supports bookmarks for value-sensitive and insensitive cursors. For example, the ODBC cursor types `SQL_CURSOR_STATIC` and `SQL_CURSOR_KEYSET_DRIVEN` support bookmarks while cursor types `SQL_CURSOR_DYNAMIC` and `SQL_CURSOR_FORWARD_ONLY` do not.

Before ODBC 3.0, a database could specify only whether it supported bookmarks or not: there was no interface to provide this information for each cursor type. There was no way for a database server to indicate for what kind of cursor bookmarks were supported. For ODBC 2

applications, SAP Sybase IQ returns that it does support bookmarks. There is therefore nothing to prevent you from trying to use bookmarks with dynamic cursors; however, you should not use this combination.

Stored procedure considerations

This section describes how to create and call stored procedures and process the results from an ODBC application.

Procedures and result sets

There are two types of procedures: those that return result sets and those that do not. You can use `SQLNumResultCols` to tell the difference: the number of result columns is zero if the procedure does not return a result set. If there is a result set, you can fetch the values using `SQLFetch` or `SQLExtendedFetch` just like any other cursor.

Parameters to procedures should be passed using parameter markers (question marks). Use `SQLBindParameter` to assign a storage area for each parameter marker, whether it is an INPUT, OUTPUT, or INOUT parameter.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure-defined result set. Therefore, ODBC does not always describe column names as defined in the `RESULT` clause of the stored procedure definition. To avoid this problem, you can use column aliases in your procedure result set cursor.

Example 1

This example creates and calls a procedure that does not return a result set. The procedure takes one INOUT parameter, and increments its value. In the example, the variable `num_columns` has the value zero, since the procedure does not return a result set. Error checking has been omitted to make the example easier to read.

```
HDBC dbc;
SQLHSTMT stmt;
SQLINTEGER I;
SQLSMALLINT num_columns;

SQLAllocStmt( dbc, &stmt );
SQLExecDirect( stmt,
    "CREATE PROCEDURE Increment( INOUT a INT )"
    "BEGIN "
    "    SET a = a + 1 "
    "END", SQL_NTS );

/* Call the procedure to increment 'I' */
I = 1;
SQLBindParameter( stmt, 1, SQL_C_LONG, SQL_INTEGER, 0, 0, &I, NULL );
SQLExecDirect( stmt, "CALL Increment( ? )", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
```

Example 2

This example calls a procedure that returns a result set. In the example, the variable `num_columns` will have the value 2 since the procedure returns a result set with two columns. Again, error checking has been omitted to make the example easier to read.

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT num_columns;

SQLCHAR ID[ 10 ];
SQLCHAR Surname[ 20 ];

SQLExecDirect( stmt,
    "CREATE PROCEDURE EmployeeList() "
    "RESULT( ID CHAR(10), Surname CHAR(20) ) "
    "BEGIN "
    "    SELECT EmployeeID, Surname FROM Employees "
    "END", SQL_NTS );

/* Call the procedure - print the results */
SQLExecDirect( stmt, "CALL EmployeeList()", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
SQLBindCol( stmt, 1, SQL_C_CHAR, &ID, sizeof(ID), NULL );
SQLBindCol( stmt, 2, SQL_C_CHAR, &Surname, sizeof(Surname), NULL );

for( ;; )
{
    rc = SQLFetch( stmt );
    if( rc == SQL_NO_DATA_FOUND )
    {
        rc = SQLMoreResults( stmt );
        if( rc == SQL_NO_DATA_FOUND ) break;
    }
    else
    {
        do_something( ID, Surname );
    }
}
```

ODBC escape syntax

You can use ODBC escape syntax from any ODBC application. This escape syntax allows you to call a set of common functions regardless of the database management system you are using. The general form for the escape syntax is

```
{ keyword parameters }
```

The set of keywords includes the following:

- **{d date-string}** – The date string is any date value accepted by SAP Sybase IQ.

- **{t time-string}** – The time string is any time value accepted by SAP Sybase IQ.
- **{ts date-string time-string}** – The date/time string is any timestamp value accepted by SAP Sybase IQ.
- **{guid uuid-string}** – The uuid-string is any valid GUID string, for example, 41dfe9ef-db91-11d2-8c43-006008d26a6f.
- **{oj outer-join-expr}** – The outer-join-expr is a valid OUTER JOIN expression accepted by SAP Sybase IQ.
- **{? = call func(p1,...)}** – The function is any valid function call accepted by SAP Sybase IQ.
- **{call proc(p1,...)}** – The procedure is any valid stored procedure call accepted by SAP Sybase IQ.
- **{fn func(p1,...)}** – The function is any one of the library of functions listed below.

You can use the escape syntax to access a library of functions implemented by the ODBC driver that includes number, string, time, date, and system functions.

For example, to obtain the current date in a database management system-neutral way, you would execute the following:

```
SELECT { FN CURDATE ( ) }
```

The following tables list the functions that are supported by the SAP Sybase IQ ODBC driver.

SAP Sybase IQ ODBC driver supported functions

Numeric functions	String functions	System functions	Time/date functions
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH	CONVERT	CURRENT_TIME-STAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT

Numeric functions	String functions	System functions	Time/date functions
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		TIMESTAMPADD
ROUND	RTRIM		TIMESTAMPDIFF
SIGN	SOUNDEX		WEEK
SIN	SPACE		YEAR
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

ODBC TIMESTAMPADD, TIMESTAMPDIFF

The ODBC driver maps the TIMESTAMPADD and TIMESTAMPDIFF functions to the corresponding database server DATEADD and DATEDIFF functions. The syntax for the TIMESTAMPADD and TIMESTAMPDIFF functions is as follows.

```
{ fn TIMESTAMPADD( interval, integer-expr, timestamp-expr ) }
```

Returns the timestamp calculated by adding *integer-expr* intervals of type *interval* to *timestamp-expr*. Valid values of *interval* are shown below.

```
{ fn TIMESTAMPDIFF( interval, timestamp-expr1, timestamp-expr2 ) }
```

Returns the integer number of intervals of type *interval* by which *timestamp-expr2* is greater than *timestamp-expr1*. Valid values of *interval* are shown below.

interval	DATEADD/DATE-DIFF date-part mapping
SQL_TSI_YEAR	YEAR
SQL_TSI_QUARTER	QUARTER

interval	DATEADD/DATE-DIFF date-part mapping
SQL_TSI_MONTH	MONTH
SQL_TSI_WEEK	WEEK
SQL_TSI_DAY	DAY
SQL_TSI_HOUR	HOUR
SQL_TSI_MINUTE	MINUTE
SQL_TSI_SECOND	SECOND
SQL_TSI_FRAC_SECOND	MICROSECOND - The DATEADD and DATEDIFF functions do not support a resolution of nanoseconds.

Interactive SQL

The ODBC escape syntax is identical to the JDBC escape syntax. In Interactive SQL, which uses JDBC, the braces *must* be doubled. There must not be a space between successive braces: "{" is acceptable, but "{" " is not. As well, you cannot use newline characters in the statement. The escape syntax cannot be used in stored procedures because they are not parsed by Interactive SQL.

For example, to obtain the number of weeks in February 2013, execute the following in Interactive SQL:

```
SELECT {{ fn TIMESTAMPDIFF(SQL_TSI_WEEK, '2013-02-01T00:00:00',
'2013-03-01T00:00:00' ) }}
```

Error handling in ODBC

Errors in ODBC are reported using the return value from each of the ODBC function calls and either the `SQLERROR` function or the `SQLGetDiagRec` function. The `SQLERROR` function was used in ODBC versions up to, but not including, version 3. As of version 3 the `SQLERROR` function has been deprecated and replaced by the `SQLGetDiagRec` function.

Every ODBC function returns a `SQLRETURN`, which is one of the following status codes:

Status code	Description
SQL_SUCCESS	No error.

Status code	Description
SQL_SUCCESS_WITH_INFO	<p>The function completed, but a call to <code>SQLError</code> will indicate a warning.</p> <p>The most common case for this status is that a value being returned is too long for the buffer provided by the application.</p>
SQL_ERROR	<p>The function did not complete because of an error. Call <code>SQLError</code> to get more information about the problem.</p>
SQL_INVALID_HANDLE	<p>An invalid environment, connection, or statement handle was passed as a parameter.</p> <p>This often happens if a handle is used after it has been freed, or if the handle is the null pointer.</p>
SQL_NO_DATA_FOUND	<p>There is no information available.</p> <p>The most common use for this status is when fetching from a cursor; it indicates that there are no more rows in the cursor.</p>
SQL_NEED_DATA	<p>Data is needed for a parameter.</p> <p>This is an advanced feature described in the ODBC SDK documentation under <code>SQLParamData</code> and <code>SQLPutData</code>.</p>

Every environment, connection, and statement handle can have one or more errors or warnings associated with it. Each call to `SQLError` or `SQLGetDiagRec` returns the information for one error and removes the information for that error. If you do not call `SQLError` or `SQLGetDiagRec` to remove all errors, the errors are removed on the next function call that passes the same handle as a parameter.

Each call to `SQLError` passes three handles for an environment, connection, and statement. The first call uses `SQL_NULL_HSTMT` to get the error associated with a connection. Similarly, a call with both `SQL_NULL_DBC` and `SQL_NULL_HSTMT` get any error associated with the environment handle.

Each call to `SQLGetDiagRec` can pass either an environment, connection or statement handle. The first call passes in a handle of type `SQL_HANDLE_DBC` to get the error associated with a connection. The second call passes in a handle of type `SQL_HANDLE_STMT` to get the error associated with the statement that was just executed.

`SQLError` and `SQLGetDiagRec` return `SQL_SUCCESS` if there is an error to report (*not* `SQL_ERROR`), and `SQL_NO_DATA_FOUND` if there are no more errors to report.

Example 1

The following code fragment uses `SQLError` and return codes:


```

SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
UCHAR errmsg[100];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
    SQLError( env, dbc, SQL_NULL_HSTMT, NULL, NULL,
             errmsg, sizeof(errmsg), NULL );
    print_error( "Allocation failed", errmsg );
    return;
}

/* Delete items for order 2015 */
rc = SQLExecDirect( stmt,
                   "DELETE FROM SalesOrderItems WHERE ID=2015",
                   SQL_NTS );
if( rc == SQL_ERROR )
{
    SQLError( env, dbc, stmt, NULL, NULL,
             errmsg, sizeof(errmsg), NULL );
    print_error( "Failed to delete items", errmsg );
    return;
}

```

Example 2

The following code fragment uses `SQLGetDiagRec` and return codes:

```

SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT errmsglen;
SQLINTEGER errnative;
SQLCHAR errmsg[255];
SQLCHAR errstate[5];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
    SQLGetDiagRec( SQL_HANDLE_DBC, dbc, 1, errstate,
                  &errnative, errmsg, sizeof(errmsg), &errmsglen );
    print_error( "Allocation failed", errstate, errnative, errmsg );
    return;
}

rc = SQLExecDirect( stmt,
                   "DELETE FROM SalesOrderItems WHERE ID=2015",
                   SQL_NTS );
if( rc == SQL_ERROR )
{
    SQLGetDiagRec( SQL_HANDLE_STMT, stmt, 1, errstate,
                  &errnative, errmsg, sizeof(errmsg), &errmsglen );
    print_error( "Failed to delete items", errstate, errnative,
                errmsg );
}

```

ODBC CLI

```
    return;  
}
```

Java in the Database

SAP Sybase IQ provides a mechanism for executing Java classes from within the database server environment. Using Java methods in the database server provides powerful ways of adding programming logic to a database.

Java support in the database offers the following:

- Reuse Java components in the different layers of your application—client, middle-tier, or server—and use them wherever it makes the most sense to you. SAP Sybase IQ becomes a platform for distributed computing.
- Java provides a more powerful language than the SQL stored procedure language for building logic into the database.
- Java can be used in the database server without jeopardizing the integrity, security, or robustness of the database and the server.

The SQLJ standard

Java in the database is based on the SQLJ Part 1 proposed standard (ANSI/INCITS 331.1-1999). SQLJ Part 1 provides specifications for calling Java static methods as SQL stored procedures and functions.

Java in the Database FAQ

This section describes the key features of Java in the database.

What Are the Key Features of Java in the Database?

Detailed explanations of all the following points appear in later sections.

- **You can run Java in the database** – An external Java VM runs Java code on behalf of the database server.
- **You can access data from Java** – SAP Sybase IQ lets you access data from Java.
- **SQL is preserved** – The use of Java does not alter the behavior of existing SQL statements or other aspects of non-Java relational database behavior.

How Can I Use My Own Java Classes in Databases?

The Java language is more powerful than SQL. Java is an object-oriented language, so its instructions (source code) come in the form of classes. To execute Java in a database, you write the Java instructions outside the database and compile them outside the database into compiled classes (*byte code*), which are binary files holding Java instructions.

Compiled classes can be called from client applications as easily and in the same way as stored procedures. Java classes can contain both information about the subject and some

computational logic. For example, you could design, write, and compile Java code to create an Employees class complete with various methods that perform operations on an Employees table. You install your Java classes as objects into a database and write SQL cover functions or procedures to invoke the methods in the Java classes.

Once installed, you can execute these classes from the database server using stored procedures. For example, the following statement creates the interface to a Java procedure:

```
CREATE PROCEDURE MyMethod()  
EXTERNAL NAME 'JDBCExample.MyMethod()' V'  
LANGUAGE JAVA;
```

SAP Sybase IQ facilitates a runtime environment for Java classes, not a Java development environment. You need a Java development environment, such as the Java Development Kit (JDK), to write and compile Java. You also need a Java Runtime Environment to execute Java classes.

You can use many of the classes that are part of the Java API as included in the Java Development Kit. You can also use classes created and compiled by Java developers.

How Does Java Get Executed in a Database?

SAP Sybase IQ launches a Java VM. The Java VM interprets compiled Java instructions and runs them on behalf of the database server. The database server starts the Java VM automatically when needed: you do not have to take any explicit action to start or stop the Java VM.

The SQL request processor in the database server has been extended so it can call into the Java VM to execute Java instructions. It can also process requests from the Java VM to enable data access from Java.

Java Error Handling

Errors in Java applications generate an exception object representing the error (called *throwing an exception*). A thrown exception terminates a Java program unless it is caught and handled properly at some level of the application.

Both Java API classes and custom-created classes can throw exceptions. In fact, users can create their own exception classes that throw their own custom-created classes of errors.

If there is no exception handler in the body of the method where the exception occurred, then the search for an exception handler continues up the call stack. If the top of the call stack is reached and no exception handler has been found, the default exception handler of the Java interpreter running the application is called and the program terminates.

In SAP Sybase IQ, if a SQL statement calls a Java method, and an unhandled exception is thrown, a SQL error is generated. The full text of the Java exception plus the Java stack trace is displayed in the server messages window.

How to Install Java Classes into a Database

You can install Java classes into a database as a single class or a JAR.

- **A single class** – You can install a single class into a database from a compiled class file. Class files typically have extension `.class`.
- **A JAR** – You can install a set of classes all at once if they are in either a compressed or uncompressed JAR file. JAR files typically have the extension `.jar` or `.zip`. SAP Sybase IQ supports all compressed JAR files created with the JAR utility, and some other JAR compression schemes.

Class File Creation

Although the details of each step may differ depending on whether you are using a Java development tool, the steps involved in creating your own class generally include the following steps:

1. Define your class.

Write the Java code that defines your class.

2. Name and save your class.

Save your class declaration (Java code) in a file with the extension `.java`. Make certain the name of the file is the same as the name of the class and that the case of both names is identical.

For example, a class called `Utility` should be saved in a file called `Utility.java`.

3. Compile your class.

This step turns your class declaration containing Java code into a new, separate file containing byte code. The name of the new file is the same as the Java code file, but has an extension of `.class`. You can run a compiled Java class in a Java Runtime Environment, regardless of the platform you compiled it on or the operating system of the runtime environment.

Special Features of Java Classes in the Database

This section describes features of Java classes when used in the database.

How to Call the Main Method

You typically start Java applications (outside the database) by running the Java VM on a class that has a main method.

For example, the `Invoice` class in the file `%ALLUSERSPROFILE%\SybaseIQ\samples\JavaInvoice\Invoice.java` has a main method. When you execute the class from the command line using a command such as the following, it is the main method that executes.

```
java Invoice
```

Threads in Java Applications

With features of the `java.lang.Thread` package, you can use multiple threads in a Java application.

You can synchronize, suspend, resume, interrupt, or stop threads in Java applications.

No Such Method Exception

If you supply an incorrect number of arguments when calling a Java method, or if you use an incorrect data type, the Java VM responds with a

`java.lang.NoSuchMethodException` error. Check the number and type of arguments.

How to Return Result Sets from Java Methods

Write a Java method that returns a result set to the calling environment, and wrap this method in a SQL stored procedure declared to be `EXTERNAL NAME` of `LANGUAGE JAVA`.

Perform the following tasks to return result sets from a Java method:

1. Ensure that the Java method is declared as `public` and `static` in a public class.
2. For each result set you expect the method to return, ensure that the method has a parameter of type `java.sql.ResultSet[]`. These result set parameters must all occur at the end of the parameter list.
3. In the method, first create an instance of `java.sql.ResultSet` and then assign it to one of the `ResultSet[]` parameters.
4. Create a SQL stored procedure of type `EXTERNAL NAME LANGUAGE JAVA`. This type of procedure is a wrapper around a Java method. You can use a cursor on the SQL procedure result set in the same way as any other procedure that returns result sets.

Example

The following simple class has a single method that executes a query and passes the result set back to the calling environment.

```
import java.sql.*;

public class MyResultSet
{
    public static void return_rset( ResultSet[] rset1 )
        throws SQLException
    {
        Connection conn = DriverManager.getConnection(
            "jdbc:default:connection" );
        Statement stmt = conn.createStatement();
        ResultSet rset =
            stmt.executeQuery (
                "SELECT Surname " +
                "FROM Customers" );
        rset1[0] = rset;
    }
}
```

```
}
}
```

You can expose the result set using a `CREATE PROCEDURE` statement that indicates the number of result sets returned from the procedure and the signature of the Java method.

A `CREATE PROCEDURE` statement indicating a result set could be defined as follows:

```
CREATE PROCEDURE result_set()
  RESULT (SurName person_name_t)
  DYNAMIC RESULT SETS 1
  EXTERNAL NAME
    'MyResultSet.return_rset([Ljava/sql/ResultSet;)V'
  LANGUAGE JAVA;
```

You can open a cursor on this procedure, just as you can with any SAP Sybase IQ procedure returning result sets.

The string `([Ljava/sql/ResultSet;)V` is a Java method signature that is a compact character representation of the number and type of the parameters and return value.

Values Returned from Java Via Stored Procedures

You can use stored procedures created using the `EXTERNAL NAME LANGUAGE JAVA` as wrappers around Java methods. This section describes how to write your Java method to exploit `OUT` or `INOUT` parameters in the stored procedure.

Java does not have explicit support for `INOUT` or `OUT` parameters. Instead, you can use an array of the parameter. For example, to use an integer `OUT` parameter, create an array of exactly one integer:

```
public class Invoice
{
  public static boolean testOut( int[] param )
  {
    param[0] = 123;
    return true;
  }
}
```

The following procedure uses the `testOut` method:

```
CREATE PROCEDURE testOut( OUT p INTEGER )
  EXTERNAL NAME 'Invoice.testOut([I]Z'
  LANGUAGE JAVA;
```

The string `([I]Z` is a Java method signature, indicating that the method has a single parameter, which is an array of integers, and returns a Boolean value. Define the method so that the method parameter you want to use as an `OUT` or `INOUT` parameter is an array of a Java data type that corresponds to the SQL data type of the `OUT` or `INOUT` parameter.

To test this, call the stored procedure with an uninitialized variable.

```
CREATE VARIABLE zap INTEGER;  
CALL testOut( zap );  
SELECT zap;
```

The result set is 123.

Security Management for Java

Java provides security managers that you can use to control user access to security-sensitive features of your applications, such as file access and network access. You should take advantage of the security management features supported by your Java VM.

How to Start and Stop the Java VM

The Java VM loads automatically whenever the first Java operation is carried out. To load it explicitly in readiness for carrying out Java operations, you can do so by executing the following statement:

```
START JAVA;
```

You can unload the Java VM when Java is not in use using the STOP JAVA statement. The syntax is:

```
STOP JAVA;
```

Shutdown Hooks in the Java VM

The SAP Sybase IQ Java VM ClassLoader which is used in providing JAVA in the database support allows applications to install shutdown hooks. These shutdown hooks are similar to the shutdown hooks that applications install with the JVM Runtime.

When a connection that is using JAVA in the database support executes a STOP JAVA statement or disconnects, the ClassLoader for that connection runs all shutdown hooks that have been installed for that particular connection prior to unloading. For regular JAVA in the database applications that install all Java classes within the database, the installation of shutdown hooks should not be necessary. The ClassLoader shutdown hooks should be used with extreme caution and should only be used to clean up any system-wide resources that were allocated for the particular connection that is stopping Java. Also, jdbc:default JDBC requests are not allowed within shutdown hooks since the jdbc:default connection is already closed prior to the ClassLoader shutdown hook being called.

To install a shutdown hook with the SQL Anywhere Java VM ClassLoader, an application must include `sajvm.jar` in the Java compiler classpath and it needs to execute code similar to the following:

```
SDHookThread hook = new SDHookThread( ... );  
ClassLoader classLoader =  
Thread.currentThread().getContextClassLoader();
```



```
((iAnywhere.sa.jvm.SAClassLoader)classLoader).addShutdownHook( hook );
```

The `SDHookThread` class extends the standard `Thread` class and that the above code must be executed by a class that was loaded by the `ClassLoader` for the current connection. Any class that is installed within the database and that is later called via an external environment call is automatically executed by the correct SQL Anywhere Java VM `ClassLoader`.

To remove a shutdown hook from the SQL Anywhere Java VM `ClassLoader` list, an application will need to execute code similar to the following:

```
ClassLoader classLoader =  
Thread.currentThread().getContextClassLoader();  
((iAnywhere.sa.jvm.SAClassLoader)classLoader).removeShutdownHook( hook );
```

The above code must be executed by a class that was loaded by the `ClassLoader` for the current connection.

JDBC CLI

JDBC is a call-level interface for Java applications. JDBC provides you with a uniform interface to a wide range of relational databases, and provides a common base on which higher level tools and interfaces can be built. JDBC is now a standard part of Java and is included in the JDK.

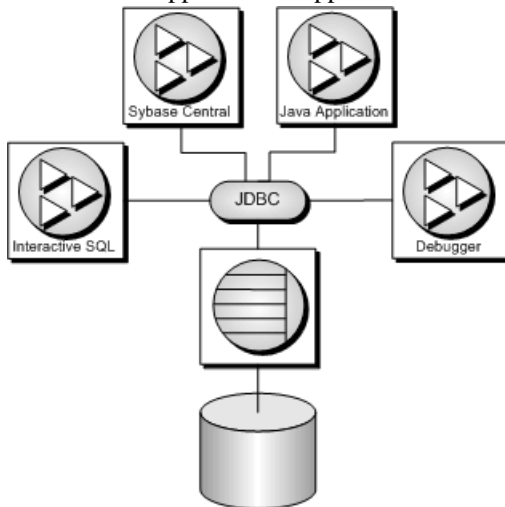
SAP Sybase IQ includes a 4.0 driver, which is a Type 2 driver.

SAP Sybase IQ also supports a pure Java JDBC driver, named jConnect, which is available from SAP.

In addition to using JDBC as a client-side application programming interface, you can also use JDBC inside the database server to access data by using Java in the database.

JDBC Applications

You can develop Java applications that use the JDBC API to connect to SAP Sybase IQ. Several of the applications supplied with SAP Sybase IQ use JDBC, such as Interactive SQL.



Java and JDBC are also important programming languages for developing UltraLite® applications.

JDBC can be used both from client applications and inside the database. Java classes using JDBC provide a more powerful alternative to SQL stored procedures for incorporating programming logic into the database.

JDBC provides a SQL interface for Java applications: to access relational data from Java, you do so using JDBC calls.

The phrase *client application* applies both to applications running on a user's computer and to logic running on a middle-tier application server.

The examples illustrate the distinctive features of using JDBC in SAP Sybase IQ. For more information about JDBC programming, see any JDBC programming book.

You can use JDBC with SAP Sybase IQ in the following ways:

- **JDBC on the client** – Java client applications can make JDBC calls to SAP Sybase IQ. The connection takes place through a JDBC driver.

SAP Sybase IQ includes a JDBC 4.0 driver, which is a Type 2 JDBC driver, and also supports the jConnect driver for pure Java applications, which is a Type 4 JDBC driver.

- **JDBC in the database** – Java classes installed into a database can make JDBC calls to access and modify data in the database using an internal JDBC driver.

JDBC resources

- **Example source code** – You can find source code for the examples in this section in the directory %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC.
- **JDBC Specification** – You can find more information about the JDBC Data Access API at <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html>.
- **Required software** – You need TCP/IP to use the jConnect driver.

The jConnect driver is available at <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>.

JDBC Drivers

SAP Sybase IQ supports the following JDBC drivers:

- **SQL Anywhere 16 JDBC 4.0 driver** – This driver communicates with SAP Sybase IQ using the Command Sequence client/server protocol. Its behavior is consistent with ODBC, embedded SQL, and OLE DB applications. The SQL Anywhere 16 JDBC 4.0 driver is the recommended JDBC driver for connecting to SAP Sybase IQ databases. The JDBC 4.0 driver can be used only with JRE 1.6 or later.

The JDBC 4.0 driver takes advantage of the new automatic JDBC driver registration. Hence, if an application wants to make use of the JDBC 4.0 driver, it no longer needs to perform a `Class.forName` call to get the JDBC driver loaded. It is instead sufficient to have the `sajdbc4.jar` file in the class file path and simply call `DriverManager.getConnection()` with a URL that begins with `jdbc:sqlanywhere`.

The JDBC 4.0 driver contains manifest information to allow it to be loaded as an OSGi (Open Services Gateway initiative) bundle.

With the JDBC 4.0 driver, metadata for NCHAR data now returns the column type as `java.sql.Types.NCHAR`, `NVARCHAR`, or `LONGNVARCHAR`. In addition, applications can now fetch NCHAR data using the `Get/SetNString` or `Get/SetNClob` methods instead of the `Get/SetString` and `Get/SetClob` methods.

- **jConnect** – This driver is a 100% pure Java driver. It communicates with SAP Sybase IQ using the TDS client/server protocol.

jConnect and jConnect documentation are available at <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>.

When choosing which driver to use, you should consider the following factors:

- **Features** – The SQL Anywhere 16 JDBC 4.0 driver and jConnect are JDBC 4.0 compliant. The SQL Anywhere 16 JDBC driver provides fully-scrollable cursors when connected to an SAP Sybase IQ database. The jConnect JDBC driver provides scrollable cursors when connected to an SAP Sybase IQ database server, but the result set is cached on the client side. The jConnect JDBC driver provides fully-scrollable cursors when connected to a SAP Adaptive Server® Enterprise database.

The JDBC 4.0 API documentation is available at <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html>.

- **Pure Java** – The jConnect driver is a pure Java solution. The SQL Anywhere 16 JDBC drivers are based on the SQL Anywhere 16 ODBC driver and are not pure Java solutions.
- **Performance** – The SQL Anywhere 16 JDBC drivers provide better performance for most purposes than the jConnect driver.
- **Compatibility** – The TDS protocol used by the jConnect driver is shared with Adaptive Server. Some aspects of the driver's behavior are governed by this protocol, and are configured to be compatible with Adaptive Server.

For information about platform availability for the SQL Anywhere 16 JDBC drivers and jConnect, see <http://www.sybase.com/detail?id=1061806>.

JDBC Program Structure

The following sequence of events typically occurs in JDBC applications:

- **Create a Connection object** – Calling a `getConnection` class method of the `DriverManager` class creates a `Connection` object, and establishes a connection with a database.
- **Generate a Statement object** – The `Connection` object generates a `Statement` object.
- **Pass a SQL statement** – A SQL statement that executes within the database environment is passed to the `Statement` object. If the statement is a query, this action returns a `ResultSet` object.

The `ResultSet` object contains the data returned from the SQL statement, but exposes it one row at a time (similar to the way a cursor works).

- **Loop over the rows of the result set** – The next method of the `ResultSet` object performs two actions:
 - The current row (the row in the result set exposed through the `ResultSet` object) advances one row.
 - A boolean value returns to indicate whether there is a row to advance to.
- **For each row, retrieve the values** – Values are retrieved for each column in the `ResultSet` object by identifying either the name or position of the column. You can use the `getData` method to get the value from a column on the current row.

Java objects can use JDBC objects to interact with a database and get data for their own use.

Differences Between Client- and Server-Side JDBC Connections

A difference between JDBC on the client and in the database server lies in establishing a connection with the database environment.

- **Client side** – In client-side JDBC, establishing a connection requires a SQL Anywhere JDBC driver or the `jConnect` JDBC driver. Passing arguments to `DriverManager.getConnection` establishes the connection. The database environment is an external application from the perspective of the client application.
- **Server-side** – When using JDBC within the database server, a connection already exists. The string `"jdbc:default:connection"` is passed to `DriverManager.getConnection`, which allows the JDBC application to work within the current user connection. This is a quick, efficient, and safe operation because the client application has already passed the database security to establish the connection. The user ID and password, having been provided once, do not need to be provided again. The server-side JDBC driver can only connect to the database of the current connection.

You can write JDBC classes so that they can run both at the client and at the server by employing a single conditional statement for constructing the URL. An external connection requires the host name and port number, while the internal connection requires `"jdbc:default:connection"`.

SQL Anywhere JDBC Drivers

The SQL Anywhere JDBC 4.0 driver provides some performance benefits and feature benefits compared to the pure Java `jConnect` JDBC driver, however, this driver does not provide a pure-Java solution. The SQL Anywhere JDBC 4.0 driver is recommended.

How to Load the SQL Anywhere JDBC 4.0 Driver

Ensure that the SQL Anywhere JDBC 4.0 driver is in your class file path.

```
set classpath=%IQDIR%\java\sajdbc4.jar;%classpath%
```

The JDBC 4.0 driver takes advantage of the new automatic JDBC driver registration. The driver is automatically loaded at execution startup when it is in the class file path.

Required Files

The Java component of the SQL Anywhere JDBC 4.0 driver is included in the `sajdbc4.jar` file installed into the `Java` subdirectory of your SAP Sybase IQ installation. For Windows, the native component is `dbjdbc16.dll` in the `bin32` or `bin64` subdirectory of your SAP Sybase IQ installation; for Unix, the native component is `libdbjdbc16.so`. This component must be in the system path.

SQL Anywhere 16 JDBC Driver Connection Strings

To connect to a database via a SQL Anywhere 16 JDBC driver, you need to supply a URL for the database. For example:

```
Connection con = DriverManager.getConnection(
    "jdbc:sqlanywhere:DSN=Sybase IQ Demo" );
```

```
Connection con =
DriverManager.getConnection("jdbc:sqlanywhere:DSN=Sybase IQ Demo" );
```

The URL contains `jdbc:sqlanywhere:` followed by a connection string. If the `sajdbc4.jar` file is in your class file path, then the JDBC 4.0 driver will have been loaded automatically and it will handle the URL. As shown in the example, an ODBC data source (DSN) may be specified for convenience, but you can also use explicit connection parameters, separated by semicolons, in addition to or instead of the data source connection parameter.

If you do not use a data source, you must specify all required connection parameters in the connection string:

```
Connection con = DriverManager.getConnection(
    "jdbc:sqlanywhere:UserID=<user_id>;Password=<password>;Start=..." );
```

The `Driver` connection parameter is not required since neither the ODBC driver nor ODBC driver manager is used. If present, it will be ignored.

The jConnect JDBC Driver

The jConnect driver is available as a separate download. To use JDBC from an applet, you must use the jConnect JDBC driver to connect to SAP Sybase IQ databases.

Download the jConnect driver at <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>. Documentation for jConnect can also be found on the same page.

The jConnect Driver Files

jConnect is supplied as a JAR file named `jconn4.jar`. This file is located in your jConnect install location.

Setting the Class File Path for jConnect

For your application to use jConnect, the jConnect classes must be in your class file path at compile time and run time, so that the Java compiler and Java runtime can locate the necessary files.

The following command adds the jConnect driver to an existing CLASSPATH environment variable (where *jconnect-path* is your jConnect installation directory).

```
set classpath=jconnect-path\classes\jconn4.jar;%classpath%
```

Importing the jConnect Classes

The classes in jConnect are all in `com.sybase.jdbc4.jdbc`. You must import these classes at the beginning of each source file:

```
import com.sybase.jdbc4.jdbc.*
```

Encrypting Passwords

SAP Sybase IQ supports password encryption for jConnect connections.

Installing jConnect System Objects into a Database

To use jConnect to access system table information (database metadata), you must add the jConnect system objects to your database.

Prerequisites

You must have the ALTER DATABASE system privilege, and must be the only connection to the database.

Back up your database files before upgrading. If you attempt to upgrade a database and it fails, then the database becomes unusable.

Task

jConnect system objects are installed into an SAP Sybase IQ database by default when you use the `iqinit` utility. You can add the jConnect system objects to the database when creating the database or at a later time by upgrading the database.

How to Load the jConnect Driver

Ensure that the jConnect driver is in your class file path. The driver file `jconn4.jar` is located in the `classes` subdirectory of your jConnect installation.

```
set classpath=.;c:\jConnect-7_0\classes\jconn4.jar;%classpath%
```

The jConnect driver takes advantage of the new automatic JDBC driver registration. The driver is automatically loaded at execution startup when it is in the class file path.

jConnect Driver Connection Strings

To connect to a database via jConnect, you need to supply a URL for the database. For example:

```
Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", "<user_id>", "<password>");
```

The URL is composed in the following way:

```
jdbc:sybase:Tds:host:port
```

The individual components are:

- **jdbc:sybase:Tds** – The jConnect JDBC driver, using the TDS application protocol.
- **host** – The IP address or name of the computer on which the server is running. If you are establishing a same-host connection, you can use `localhost`, which means the computer system you are logged into.
- **port** – The port number on which the database server listens. The port number assigned to SAP Sybase IQ is 2638. Use that number unless there are specific reasons not to do so.

The connection string must be less than 253 characters in length.

If you are using the SAP Sybase IQ personal server, make sure to include the TCP/IP support option when starting the server.

How to Specify a Database with a jConnect Connection String

Each SAP Sybase IQ database server can have one or more databases loaded at a time. If the URL you supply when connecting via jConnect specifies a server, but does not specify a database, then the connection attempt is made to the default database on the server.

You can specify a particular database by providing an extended form of the URL in one of the following ways.

Using the ServiceName Parameter

```
jdbc:sybase:Tds:host:port?ServiceName=database
```

The question mark followed by a series of assignments is a standard way of providing arguments to a URL. The case of `ServiceName` is not significant, and there must be no spaces around the = sign. The `database` parameter is the database name, not the server name. The database name must not include the path or file suffix. For example:

```
Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638?ServiceName=demo", "DBA",
    "sql");
```

Using the RemotePWD Parameter

A workaround exists for passing additional connection parameters to the server.

This technique allows you to provide additional connection parameters such as the database name, or a database file, using the `RemotePWD` field. You set `RemotePWD` as a Properties field using the `put` method.

The following code illustrates how to use the field.

```
import java.util.Properties;
.
.
.
Properties props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "sql" );
props.put( "RemotePWD", ",DatabaseFile=mydb.db" );

Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", props );
```

As shown in the example, a comma must precede the `DatabaseFile` connection parameter. Using the `DatabaseFile` parameter, you can start a database on a server using `jConnect`. By default, the database is started with `AutoStop=YES`. If you specify `utility_db` with a `DatabaseFile` (DBF) or `DatabaseName` (DBN) connection parameter (for example, `DBN=utility_db`), then the utility database is started automatically.

Database Options Set for jConnect Connections

When an application connects to the database using the `jConnect` driver, the `sp_tsql_environment` stored procedure is called. The `sp_tsql_environment` procedure sets some database options for compatibility with Adaptive Server Enterprise behavior.

Connections from a JDBC Client Application

Database metadata is always available when using a SQL Anywhere JDBC driver.

To access database system tables (database metadata) from a JDBC application that uses `jConnect`, you must add a set of `jConnect` system objects to your database. These procedures are installed to all databases by default. The `iqinit -i` option prevents this installation.

The following complete Java application is a command line program that connects to a running database, prints a set of information to your command line, and terminates.

Establishing a connection is the first step any JDBC application must take when working with database data.

This example illustrates an external connection, which is a regular client/server connection.

Connection Example Code

The following example uses the JDBC 4.0 version of the SQL Anywhere JDBC driver by default to connect to the database. To use a different driver, you can pass in the driver name (jdbc4,jConnect) on the command line. Examples for using the JDBC 4.0 driver and jConnect are included in the code. This example assumes that a database server has already been started using the sample database. The source code can be found in the file `JDBCCConnect.java` in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` directory.

```
import java.io.*;
import java.sql.*;

public class JDBCCConnect
{
    public static void main( String args[] )
    {
        try
        {
            String arg;
            Connection con;

            // Select the JDBC driver and create a connection.
            // May throw a SQLException.
            // Choices are:
            // 1. jConnect driver
            // 2. SQL Anywhere JDBC 4.0 driver
            arg = "jdbc4";
            if( args.length > 0 ) arg = args[0];
            if( arg.compareToIgnoreCase( "jconnect" ) == 0 )
            {
                con = DriverManager.getConnection(
                    "jdbc:sybase:Tds:localhost:2638", "<user_id>",
"<password>");
            }
            else
            {
                con = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=<user_id>;pwd=<password>" );
            }

            System.out.println("Using "+arg+" driver");

            // Create a statement object, the container for the SQL
            // statement. May throw a SQLException.
            Statement stmt = con.createStatement();

```

```

// Create a result set object by executing the query.
// May throw a SQLException.
ResultSet rs = stmt.executeQuery(
    "SELECT ID, GivenName, Surname FROM Customers");

// Process the result set.
while (rs.next())
{
    int value = rs.getInt(1);
    String FirstName = rs.getString(2);
    String LastName = rs.getString(3);
    System.out.println(value+" "+FirstName+" "+LastName);
}
rs.close();
stmt.close();
con.close();
}
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());

    System.exit(1);
}
catch (Exception e)
{
    e.printStackTrace();
    System.exit(1);
}

System.exit(0);
}
}

```

How the Connection Example Works

The external connection example is a Java command line program.

Importing Packages

The application requires a couple of packages, which are imported in the first lines of `JDBCCConnect.java`:

- The `java.io` package contains the Java input/output classes, which are required for printing to the command prompt window.
- The `java.sql` package contains the JDBC classes, which are required for all JDBC applications.

The Main Method

Each Java application requires a class with a method named `main`, which is the method invoked when the program starts. In this simple example, `JDBCCConnect.main` is the only public method in the application.

The `JDBCCConnect.main` method carries out the following tasks:

1. Determines which driver to load based on the command line argument. The SQL Anywhere JDBC 4.0 and jConnect 7.0 drivers are automatically loaded at startup if they are in the class file path.
2. Connects to the default running database using the selected JDBC driver URL. The `getConnection` method establishes a connection using the specified URL.
3. Creates a statement object, which is the container for the SQL statement.
4. Creates a result set object by executing a SQL query.
5. Iterates through the result set, printing the column information.
6. Closes each of the result set, statement, and connection objects.

Running the Connection Example

The steps involved in creating and executing a JDBC application are shown through the use of an example.

Prerequisites

A Java Development Kit (JDK) must be installed.

Task

Two different types of connections using JDBC can be made. One is the client-side connection and the other is the server-side connection. The following example uses a client-side connection.

1. At a command prompt, change to the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` directory.
2. Start a database server with the `iqdemo.db` database on your local computer.
3. Set the `CLASSPATH` environment variable. The SQL Anywhere JDBC 4.0 driver contained in `sajdbc4.jar` is used in this example.

```
set classpath=.;%IQDIR%\java\sajdbc4.jar
```

If you are using the jConnect driver instead, then use the following (where *path* is your jConnect installation directory):

```
set classpath=.;jconnect-path\classes\jconn4.jar
```

4. Run the following command to compile the example:

```
javac JDBCCConnect.java
```

5. Run the following command to execute the example:

```
java JDBCCConnect
```

Add a command line argument such as `jconnect` to load a different JDBC driver.

```
java JDBCCConnect jconnect
```

6. Confirm that a list of identification numbers with customer's names appears at the command prompt.

If the attempt to connect fails, an error message appears instead. Confirm that you have executed all the steps as required. Check that your class file path is correct. An incorrect setting may result in a failure to locate a class.

A list of identification numbers with customer's names is displayed.

How to Establish a Connection from a Server-Side JDBC Class

SQL statements in JDBC are built using the `createStatement` method of a `Connection` object. Even classes running inside the server need to establish a connection to create a `Connection` object.

Establishing a connection from a server-side JDBC class is more straightforward than establishing an external connection. Because the user is already connected to the database, the class simply uses the current connection.

Server-Side Connection Example Code

The following is the source code for the server-side connection example. It is a modified version of the `JDBCCConnect.java` example and is located in `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCCConnect2.java`.

```
import java.io.*;
import java.sql.*;

public class JDBCCConnect2
{
    public static void main( String args[] )
    {
        try
        {
            // Open the connection. May throw a SQLException.
            Connection con = DriverManager.getConnection(
                "jdbc:default:connection" );

            // Create a statement object, the container for the SQL
            // statement. May throw a SQLException.
            Statement stmt = con.createStatement();
            // Create a result set object by executing the query.
            // May throw a SQLException.
            ResultSet rs = stmt.executeQuery(
                "SELECT ID, GivenName, Surname FROM Customers");

            // Process the result set.
            while (rs.next())
            {
                int value = rs.getInt(1);
                String FirstName = rs.getString(2);
                String LastName = rs.getString(3);
                System.out.println(value+" "+FirstName+" "+LastName);
            }
        }
    }
}
```

```

    }
    rs.close();
    stmt.close();
    con.close();
}
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
}
catch (Exception e)
{
    e.printStackTrace();
}
}
}

```

How the Server-Side Connection Example Differs

The server-side connection example is almost identical to the client-side connection example, with the following exceptions:

1. The JDBC driver does not need to be preloaded.
2. It connects to the default running database using the current connection. The URL in the getConnection call has been changed as follows:

```

Connection con = DriverManager.getConnection(
    "jdbc:default:connection" );

```

3. The System.exit() statements have been removed.

Running the Server-Side Connection Example

The steps involved in creating and executing a JDBC server-side application are shown through the use of an example.

Prerequisites

A Java Development Kit (JDK) must be installed.

Task

Two different types of connections using JDBC can be made. One is the client-side connection and the other is the server-side connection. The following example uses a server-side connection.

1. At a command prompt, change to the %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC directory.

```

cd %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC

```

2. For server-side JDBC, it is not necessary to set the CLASSPATH environment variable unless the server will be started from a different current working directory.

```
set classpath=.;%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere
\JDBC
```

3. Start a database server with the iqdemo database on your local computer.

4. Enter the following command to compile the example:

```
javac JDBCCConnect2.java
```

5. Install the class into the sample database using Interactive SQL. Execute the following statement (a path to the class file may be required):

```
INSTALL JAVA NEW
FROM FILE 'JDBCCConnect2.class';
```

6. Define a stored procedure named JDBCCConnect that acts as a wrapper for the JDBCCConnect2.main method in the class:

```
CREATE PROCEDURE JDBCCConnect(OUT args LONG VARCHAR)
EXTERNAL NAME 'JDBCCConnect2.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

7. Call the JDBCCConnect2.main method as follows:

```
CALL JDBCCConnect();
```

The first time a Java class is called in a session, the Java VM must be loaded. This might take a few seconds.

8. Confirm that a list of identification numbers with customers' names appears in the database server messages window.

If the attempt to connect fails, an error message appears instead. Confirm that you have executed all the steps as required.

A list of identification numbers with customer's names is displayed in the database server messages window.

Notes on JDBC Connections

Familiarize yourself with autocommit behavior, transaction isolation levels, and connection defaults.

- **Autocommit behavior** – The JDBC specification requires that, by default, a COMMIT is performed after each data manipulation statement. Currently, the client-side JDBC behavior is to commit (autocommit is true) and the server-side behavior is to not commit (autocommit is false). To obtain the same behavior in both client-side and server-side applications, you can use a statement such as the following:

```
con.setAutoCommit( false );
```

In this statement, con is the current connection object. You could also set autocommit to true.

- **Setting transaction isolation level** – To set the transaction isolation level, the application must call the Connection.setTransactionIsolation method with one of the following values.

For the SQL Anywhere JDBC 4.0 driver use:

```

TRANSACTION_NONE
TRANSACTION_READ_COMMITTED
TRANSACTION_READ_UNCOMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_SNA
PSHOT
sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_REA
ONLY_SNAPSHOT

```

The following example sets the transaction isolation level to SNAPSHOT using the JDBC 4.0 driver.

```

try
{
    con.setTransactionIsolation(
sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
    );
}
catch( Exception e )
{
    System.err.println( "Error! Could not set isolation level" );
    System.err.println( e.getMessage() );
    printExceptions( (SQLException)e );
}

```

For more information about the `getTransactionIsolation` and `setTransactionIsolation`, see documentation on the `java.sql.Connection` interface at <http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/>.

- **Connection defaults** – From server-side JDBC, only the first call to `getConnection("jdbc:default:connection")` creates a new connection with the default values. Subsequent calls return a wrapper of the current connection with all connection properties unchanged. If you set `autocommit` to `false` in your initial connection, any subsequent `getConnection` calls within the same Java code return a connection with `autocommit` set to `false`.

You may want to ensure that closing a connection restores the connection properties to their default values, so that subsequent connections are obtained with standard JDBC values. The following code achieves this:

```

Connection con =
    DriverManager.getConnection("jdbc:default:connection");

boolean oldAutoCommit = con.getAutoCommit();
try
{
    // main body of code here
}

```

```
}  
finally  
{  
    con.setAutoCommit( oldAutoCommit );  
}
```

This discussion applies not only to autocommit, but also to other connection properties such as transaction isolation level and read-only mode.

For more information about the `getTransactionIsolation`, `setTransactionIsolation`, and `isReadOnly` methods, see documentation on the `java.sql.Connection` interface at <http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/>.

Data Access Using JDBC

Java applications that hold some or all classes in the database have significant advantages over traditional SQL stored procedures. At an introductory level, however, it may be helpful to use the parallels with SQL stored procedures to demonstrate the capabilities of JDBC. In the following examples, you write Java classes that insert a row into the Departments table.

As with other interfaces, SQL statements in JDBC can be either *static* or *dynamic*. Static SQL statements are constructed in the Java application and sent to the database. The database server parses the statement, selects an execution plan, and executes the statement. Together, parsing and selecting an execution plan are referred to as *preparing* the statement.

If a similar statement has to be executed many times (many inserts into one table, for example), there can be significant overhead in static SQL because the preparation step has to be executed each time.

In contrast, a dynamic SQL statement contains place holders. The statement, prepared once using these place holders, can be executed many times without the additional expense of preparing.

Preparing for the JDBC Examples

The code fragments in the following sections are taken from the complete class in `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCExample.java`. In preparation for these sections, the sample Java application is compiled and installed into the database.

Prerequisites

You must have the `MANAGE ANY EXTERNAL OBJECT` system privilege.

A Java Development Kit (JDK) must be installed.

Task

1. Compile the `JDBCExample.java` source code.
2. Connect to the database from Interactive SQL.
3. Install the `JDBCExample.class` file into the sample database by executing the following statement in Interactive SQL:

```
INSTALL JAVA NEW
FROM FILE 'JDBCExample.class';
```

If the database server was not started from the same directory as the class file and the path to the class file is not listed in the database server's `CLASSPATH`, then you will have to include the path to the class file in the `INSTALL` statement.

The `JDBCExample` class file is installed in the database and ready for demonstration.

Inserts, Updates, and Deletes Using JDBC

Static SQL statements such as `INSERT`, `UPDATE`, and `DELETE`, which do not return result sets, are executed using the `executeUpdate` method of the `Statement` class. Statements, such as `CREATE TABLE` and other data definition statements, can also be executed using `executeUpdate`.

The `addBatch`, `clearBatch`, and `executeBatch` methods of the `Statement` class may also be used. Due to the fact that the JDBC specification is unclear on the behavior of the `executeBatch` method of the `Statement` class, the following notes should be considered when using this method with the SQL Anywhere JDBC drivers:

- Processing of the batch stops immediately upon encountering a SQL exception or result set. If processing of the batch stops, then a `BatchUpdateException` will be thrown by the `executeBatch` method. Calling the `getUpdateCounts` method on the `BatchUpdateException` will return an integer array of row counts where the set of counts prior to the batch failure will contain a valid non-negative update count; while all counts at the point of the batch failure and beyond will contain a -1 value. Casting the `BatchUpdateException` to a `SQLException` will provide additional details as to why batch processing was stopped.
- The batch is only cleared when the `clearBatch` method is explicitly called. As a result, calling the `executeBatch` method repeatedly will re-execute the batch over and over again. In addition, calling `execute(sql_query)` or `executeQuery(sql_query)` will correctly execute the specified SQL query, but will not clear the underlying batch. Hence, calling the `executeBatch` method followed by `execute(sql_query)` followed by the `executeBatch` method again will execute the set of batched statements, then execute the specified SQL query, and then execute the set of batched statements again.

The following code fragment illustrates how to execute an `INSERT` statement. It uses a `Statement` object that has been passed to the `InsertStatic` method as an argument.

```
public static void InsertStatic( Statement stmt )
{
```

```

try
{
    int iRows = stmt.executeUpdate(
        "INSERT INTO Departments (DepartmentID, DepartmentName) "
        + " VALUES (201, 'Eastern Sales')" );
    // Print the number of rows inserted
    System.out.println(iRows + " rows inserted");
}
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
}
catch (Exception e)
{
    e.printStackTrace();
}
}

```

Notes

- This code fragment is part of the `JDBCExample.java` file included in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` directory.
- The `executeUpdate` method returns an integer that reflects the number of rows affected by the operation. In this case, a successful `INSERT` would return a value of one (1).
- When run as a server-side class, the output from `System.out.println` goes to the database server messages window.

Using Static INSERT and DELETE Statements from JDBC

A sample JDBC application is called from the database server to insert and delete rows in the `Departments` table using static SQL statements.

Prerequisites

To create an external procedure, you must have the `CREATE PROCEDURE` and `CREATE EXTERNAL REFERENCE` system privileges. You must also have `SELECT`, `DELETE`, and `INSERT` privileges on the database object you are modifying.

A Java Development Kit (JDK) must be installed.

Task

1. Connect to the database from Interactive SQL.
2. Ensure the `JDBCExample` class has been installed.
3. Define a stored procedure named `JDBCExample` that acts as a wrapper for the `JDBCExample.main` method in the class:

```
CREATE PROCEDURE JDBCEXample(IN arg CHAR(50))
  EXTERNAL NAME 'JDBCEXample.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
```

4. Call the JDBCEXample.main method as follows:

```
CALL JDBCEXample( 'insert' );
```

The argument string 'insert' causes the InsertStatic method to be invoked.

5. Confirm that a row has been added to the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

6. There is a similar method in the example class called DeleteStatic that shows how to delete the row that has just been added. Call the JDBCEXample.main method as follows:

```
CALL JDBCEXample( 'delete' );
```

The argument string 'delete' causes the DeleteStatic method to be invoked.

7. Confirm that the row has been deleted from the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

Rows are inserted and deleted from a table using static SQL statements in a server-side JDBC application.

How to Use Prepared Statements for More Efficient Access

If you use the Statement interface, you parse each statement that you send to the database, generate an access plan, and execute the statement. The steps before execution are called *preparing* the statement.

You can achieve performance benefits if you use the PreparedStatement interface. This allows you to prepare a statement using placeholders, and then assign values to the placeholders when executing the statement.

Using prepared statements is particularly useful when carrying out many similar actions, such as inserting many rows.

Example

The following example illustrates how to use the PreparedStatement interface, although inserting a single row is not a good use of prepared statements.

The following InsertDynamic method of the JDBCEXample class carries out a prepared statement:

```
public static void InsertDynamic( Connection con,
                                String ID, String name )
{
```

```

try
{
    // Build the INSERT statement
    // ? is a placeholder character
    String sqlStr = "INSERT INTO Departments " +
        "( DepartmentID, DepartmentName ) " +
        "VALUES ( ? , ? )";

    // Prepare the statement
    PreparedStatement stmt =
        con.prepareStatement( sqlStr );

    // Set some values
    int idValue = Integer.valueOf( ID );
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );

    // Execute the statement
    int iRows = stmt.executeUpdate();

    // Print the number of rows inserted
    System.out.println(iRows + " rows inserted");
}
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
}
catch (Exception e)
{
    e.printStackTrace();
}
}

```

Notes

- This code fragment is part of the `JDBCExample.java` file included in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` directory.
- The `executeUpdate` method returns an integer that reflects the number of rows affected by the operation. In this case, a successful INSERT would return a value of one (1).
- When run as a server-side class, the output from `System.out.println` goes to the database server messages window.

Using Prepared INSERT and DELETE Statements from JDBC

A sample JDBC application is called from the database server to insert and delete rows in the Departments table using prepared statements.

Prerequisites

To create an external procedure, you must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges. You must also have SELECT, DELETE, and INSERT privileges on the database object you are modifying.

A Java Development Kit (JDK) must be installed.

Task

1. Connect to the database from Interactive SQL.
2. Ensure the JDBCExample class has been installed.
3. Define a stored procedure named JDBCInsert that acts as a wrapper for the JDBCExample.Insert method in the class:

```
CREATE PROCEDURE JDBCInsert(IN arg1 INTEGER, IN arg2 CHAR(50))
  EXTERNAL NAME 'JDBCExample.Insert(ILjava/lang/String;)V'
  LANGUAGE JAVA;
```

4. Call the JDBCExample.Insert method as follows:

```
CALL JDBCInsert( 202, 'Southeastern Sales' );
```

The Insert method causes the InsertDynamic method to be invoked.

5. Confirm that a row has been added to the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

6. There is a similar method in the example class called DeleteDynamic that shows how to delete the row that has just been added.

Define a stored procedure named JDBCDelete that acts as a wrapper for the JDBCExample.Delete method in the class:

```
CREATE PROCEDURE JDBCDelete(IN arg1 INTEGER)
  EXTERNAL NAME 'JDBCExample.Delete(I)V'
  LANGUAGE JAVA;
```

7. Call the JDBCExample.Delete method as follows:

```
CALL JDBCDelete( 202 );
```

The Delete method causes the DeleteDynamic method to be invoked.

8. Confirm that the row has been deleted from the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

Rows are inserted and deleted from a table using prepared SQL statements in a server-side JDBC application.

JDBC Batch Methods

The `addBatch` method of the `PreparedStatement` class is used for performing batched (or wide) inserts. The following are some guidelines to using this method.

1. An `INSERT` statement should be prepared using one of the `prepareStatement` methods of the `Connection` class.

```
// Build the INSERT statement
String sqlStr = "INSERT INTO Departments " +
               "( DepartmentID, DepartmentName ) " +
               "VALUES ( ? , ? )";
// Prepare the statement
PreparedStatement stmt =
    con.prepareStatement( sqlStr );
```

2. The parameters for the prepared insert statement should be set and batched as follows:

```
// loop to batch "n" sets of parameters
for( i=0; i < n; i++ )
{
    // "stmt" is the original prepared insert statement from step
    1.
    stmt.setSomeType( 1, param_1 );
    stmt.setSomeType( 2, param_2 );
    .
    .
    // There are "m" parameters in the statement.
    stmt.setSomeType( m , param_m );

    // Add the set of parameters to the batch and
    // move to the next row of parameters.
    stmt.addBatch();
}
```

Example:

```
for( i=0; i < 5; i++ )
{
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );
    stmt.addBatch();
}
```

3. The batch must be executed using the `executeBatch` method of the `PreparedStatement` class.

BLOB parameters are not supported in batches.

When using the SQL Anywhere JDBC driver to perform batched inserts, it is recommended that you use a small column size. Using batched inserts to insert large binary or character data

into long binary or long varchar columns is not recommended and may degrade performance. The performance can decrease because the SQL Anywhere JDBC driver must allocate large amounts of memory to hold each of the batched insert rows. In all other cases, using batched inserts should provide better performance than using individual inserts.

How to Return Result Sets from Java

This section describes how to make one or more result sets available from Java methods.

You must write a Java method that returns one or more result sets to the calling environment, and wrap this method in a SQL stored procedure. The following code fragment illustrates how multiple result sets can be returned to the caller of this Java procedure. It uses three `executeQuery` statements to obtain three different result sets.

```
public static void Results( ResultSet[] rset )
    throws SQLException
{
    // Demonstrate returning multiple result sets

    Connection con = DriverManager.getConnection(
        "jdbc:default:connection" );
    rset[0] = con.createStatement().executeQuery(
        "SELECT * FROM Employees" +
        "    ORDER BY EmployeeID" );
    rset[1] = con.createStatement().executeQuery(
        "SELECT * FROM Departments" +
        "    ORDER BY DepartmentID" );
    rset[2] = con.createStatement().executeQuery(
        "SELECT i.ID,i.LineID,i.ProductID,i.Quantity," +
        "        s.OrderDate,i.ShipDate," +
        "        s.Region,e.GivenName||' '||e.Surname" +
        "    FROM SalesOrderItems AS i" +
        "    JOIN SalesOrders AS s" +
        "    JOIN Employees AS e" +
        "    WHERE s.ID=i.ID" +
        "        AND s.SalesRepresentative=e.EmployeeID" );
    con.close();
}
```

Notes

- This server-side JDBC example is part of the `JDBCExample.java` file included in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` directory.
- It obtains a connection to the default running database by using `getConnection`.
- The `executeQuery` methods return result sets.

Returning Result Sets from JDBC

A sample JDBC application is called from the database server to return multiple result sets.

Prerequisites

A Java Development Kit (JDK) must be installed.

Task

1. Connect to the database from Interactive SQL.
2. Ensure the JDBCExample class has been installed.
3. Define a stored procedure named JDBCResults that acts as a wrapper for the JDBCExample.Results method in the class.

For example:

```
CREATE PROCEDURE JDBCResults(OUT args LONG VARCHAR)
  DYNAMIC RESULT SETS 3
  EXTERNAL NAME 'JDBCExample.Results([Ljava/sql/ResultSet;)V'
  LANGUAGE JAVA;
```

The example returns 3 result sets.

4. Set the following Interactive SQL options so you can see all the results of the query:
 - a. Click **Tools » Options**.
 - b. Click **Sybase IQ**.
 - c. Click the **Results** tab.
 - d. Set the value for **Maximum Number Of Rows To Display** to 5000.
 - e. Click **Show All Result Sets**.
 - f. Click **OK**.
5. Call the JDBCExample.Results method.

```
CALL JDBCResults();
```

6. Check each of the three results tabs, **Result Set 1**, **Result Set 2**, and **Result Set 3**.

Three different result sets are returned from a server-side JDBC application.

JDBC Notes

Learn about privileges for accessing and executing Java classes.

- **Access privileges** – Like all Java classes in the database, classes containing JDBC statements can be accessed by any user if the GRANT EXECUTE statement has granted them privilege to execute the stored procedure that is acting as a wrapper for the Java method.
- **Execution privileges** – Java classes are executed with the privileges of the connection executing them. This behavior is different from that of stored procedures, which execute with the privileges of the owner.

JDBC Callbacks

The SQL Anywhere JDBC driver supports two asynchronous callbacks, one for handling the SQL MESSAGE statement and the other for validating requests for file transfers.

Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

A message handler routine can be created to intercept these messages. The following is an example of a message handler callback routine.

```
class T_message_handler implements
sybase.jdbc4.sqlanywhere.ASAMessageHandler
{
    private final int MSG_INFO      = 0x80 | 0;
    private final int MSG_WARNING   = 0x80 | 1;
    private final int MSG_ACTION    = 0x80 | 2;
    private final int MSG_STATUS    = 0x80 | 3;
    T_message_handler()
    {
    }

    public SQLException messageHandler(SQLException sqe)
    {
        String msg_type = "unknown";

        switch( sqe.getErrorCode() ) {
            case MSG_INFO:      msg_type = "INFO "; break;
            case MSG_WARNING:   msg_type = "WARNING"; break;
            case MSG_ACTION:    msg_type = "ACTION "; break;
            case MSG_STATUS:    msg_type = "STATUS "; break;
        }

        System.out.println( msg_type + ": " + sqe.getMessage() );
        return sqe;
    }
}
```

A client file transfer request can be validated. Before allowing any transfer to take place, the JDBC driver will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the JDBC driver will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below. The following is an example of a file transfer validation callback routine.

```
class T_filetrans_callback implements
sybase.jdbc4.sqlanywhere.SAValidateFileTransferCallback
{
    T_filetrans_callback()
    {
    }
}
```

```

    public int callback(String filename, int is_write)
    {
        System.out.println( "File transfer granted for file " +
filename +
                                " with an is_write value of " +
is_write );
        return( 1 ); // 0 to disallow, non-zero to allow
    }
}

```

The filename argument is the name of the file to be read or written. The is_write parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the JDBC driver allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. If the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

The following sample Java application demonstrates the use of the callbacks supported by the SQL Anywhere JDBC 4.0 driver. You need to place the file %ALLUSERSPROFILE%\SybaseIQ\samples\java\sajdbc4.jar in your classpath.

```

import java.io.*;
import java.sql.*;
import java.util.*;

public class callback
{
    public static void main (String args[]) throws IOException
    {
        Connection        con = null;
        Statement          stmt;

        System.out.println ( "Starting... " );
        con = connect();
        if( con == null )
        {
            return; // exception should already have been reported
        }
        System.out.println ( "Connected... " );
    }
}

```

```

        try
        {
            // create and register message handler callback
            T_message_handler message_worker = new
T_message_handler();

            ((sybase.jdbc4.sqlanywhere.IConnection)con).setASAMessageHandler( m
essage_worker );

            // create and register validate file transfer callback
            T_filetrans_callback filetran_worker = new
T_filetrans_callback();

            ((sybase.jdbc4.sqlanywhere.IConnection)con).setSAValidateFileTransf
erCallback( filetran_worker );

            stmt = con.createStatement();

            // execute message statements to force message handler to
be called
            stmt.execute( "MESSAGE 'this is an info  message' TYPE
INFO TO CLIENT" );
            stmt.execute( "MESSAGE 'this is an action message' TYPE
ACTION TO CLIENT" );
            stmt.execute( "MESSAGE 'this is a warning message' TYPE
WARNING TO CLIENT" );
            stmt.execute( "MESSAGE 'this is a status  message' TYPE
STATUS TO CLIENT" );

            System.out.println( "\n=====\\n" );

            stmt.execute( "set temporary option
allow_read_client_file='on'" );
            try
            {
                stmt.execute( "drop procedure read_client_file_test" );
            }
            catch( SQLException dummy )
            {
                // ignore exception if procedure does not exist
            }
            // create procedure that will force file transfer callback
to be called
            stmt.execute( "create procedure read_client_file_test()" +
                "begin" +
                "    declare v long binary;" +
                "    set v = read_client_file('sample.txt');" +
                "end" );

            // call procedure to force validate file transfer callback
to be called
            try
            {
                stmt.execute( "call read_client_file_test()" );
            }
            catch( SQLException filetrans_exception )

```

```

        {
            // Note: Since the file transfer callback returns 1,
            // do not expect a SQL exception to be thrown
            System.out.println( "SQLException: " +
                               filetrans_exception.getMessage() );
        }
        stmt.close();
        con.close();
        System.out.println( "Disconnected" );
    }
    catch( SQLException sqe )
    {
        printExceptions(sqe);
    }
}

private static Connection connect()
{
    Connection connection;

    System.out.println( "Using jdbc4 driver" );
    try
    {
        connection = DriverManager.getConnection(
            "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
    }
    catch( Exception e )
    {
        System.err.println( "Error! Could not connect" );
        System.err.println( e.getMessage() );
        printExceptions( (SQLException)e );
        connection = null;
    }
    return connection;
}

static private void printExceptions(SQLException sqe)
{
    while (sqe != null)
    {
        System.out.println("Unexpected exception : " +
            "SqlState: " + sqe.getSQLState() +
            " " + sqe.toString() +
            ", ErrorCode: " + sqe.getErrorCode());
        System.out.println( "=====\n" );
        sqe = sqe.getNextException();
    }
}
}

```

JDBC Escape Syntax

You can use JDBC escape syntax from any JDBC application, including Interactive SQL. This escape syntax allows you to call stored procedures regardless of the database management system you are using. The general form for the escape syntax is

```
{ keyword parameters }
```

The set of keywords includes the following:

- **{d date-string}** – The date string is any date value accepted by SAP Sybase IQ.
- **{t time-string}** – The time string is any time value accepted by SAP Sybase IQ.
- **{ts date-string time-string}** – The date/time string is any timestamp value accepted by SAP Sybase IQ.
- **{guid uuid-string}** – The uuid-string is any valid GUID string, for example, 41dfe9ef-db91-11d2-8c43-006008d26a6f.
- **{oj outer-join-expr}** – The outer-join-expr is a valid OUTER JOIN expression accepted by SAP Sybase IQ.
- **{? = call func(p1,...)}** – The function is any valid function call accepted by SAP Sybase IQ.
- **{call proc(p1,...)}** – The procedure is any valid stored procedure call accepted by SAP Sybase IQ.
- **{fn func(p1,...)}** – The function is any one of the library of functions listed below.

You can use the escape syntax to access a library of functions implemented by the JDBC driver that includes number, string, time, date, and system functions.

For example, to obtain the current date in a database management system-neutral way, you would execute the following:

```
SELECT { FN CURDATE ( ) }
```

The functions that are available depend on the JDBC driver that you are using. The following tables list the functions that are supported by the SQL Anywhere JDBC driver and by the jConnect driver.

SQL Anywhere JDBC Driver Supported Functions

Numeric functions	String functions	System functions	Time/date functions
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME

Numeric functions	String functions	System functions	Time/date functions
ATAN	CHAR_LENGTH		CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOURL
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

jConnect supported functions

Numeric functions	String functions	System functions	Time/date functions
ABS	ASCII	DATABASE	CURDATE

Numeric functions	String functions	System functions	Time/date functions
ACOS	CHAR	IFNULL	CURTIME
ASIN	CONCAT	USER	DAYNAME
ATAN	DIFFERENCE	CONVERT	DAYOFMONTH
ATAN2	LCASE		DAYOFWEEK
CEILING	LENGTH		HOUR
COS	REPEAT		MINUTE
COT	RIGHT		MONTH
DEGREES	SOUNDEX		MONTHNAME
EXP	SPACE		NOW
FLOOR	SUBSTRING		QUARTER
LOG	UCASE		SECOND
LOG10			TIMESTAMPADD
PI			TIMESTAMPDIFF
POWER			YEAR
RADIANS			
RAND			
ROUND			
SIGN			
SIN			
SQRT			
TAN			

A statement using the escape syntax should work in SAP Sybase IQ, Adaptive Server Enterprise, Oracle, SQL Server, or another database management system to which you are connected.

In Interactive SQL, the braces must be doubled. There must not be a space between successive braces: "{{" is acceptable, but "{ {" is not. As well, you cannot use newline characters in the statement. The escape syntax cannot be used in stored procedures because they are not parsed by Interactive SQL.

JDBC CLI

For example, to obtain database properties with the `sa_db_info` procedure using SQL escape syntax, you would execute the following in Interactive SQL:

```
{{CALL sa_db_info( 0 ) }}
```

JDBC 4.0 API Support

All mandatory classes and methods of the JDBC 4.0 specification are supported by the SQL Anywhere JDBC driver. Some optional methods of the `java.sql.Blob` interface are not supported. These optional methods are:

```
long position( Blob pattern, long start );  
long position( byte[] pattern, long start );  
OutputStream setBinaryStream( long pos )  
int setBytes( long pos, byte[] bytes )  
int setBytes( long pos, byte[] bytes, int offset, int len );  
void truncate( long len );
```

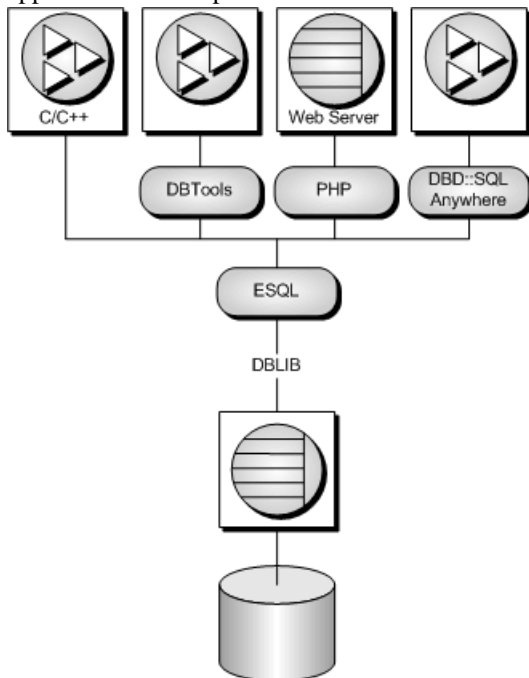
Embedded SQL

SQL statements embedded in a C or C++ source file are referred to as embedded SQL. A preprocessor translates these statements into calls to a runtime library. Embedded SQL is an ISO/ANSI and IBM standard.

Embedded SQL is portable to other databases and other environments, and is functionally equivalent in all operating environments. It is a comprehensive, low-level interface that provides all the functionality available in the product. Embedded SQL requires knowledge of C or C++ programming languages.

Embedded SQL applications

You can develop C or C++ applications that access the SAP Sybase IQ server using the SAP Sybase IQ embedded SQL interface. The command line database tools are examples of applications developed in this manner.



Embedded SQL is a database programming interface for the C and C++ programming languages. It consists of SQL statements intermixed with (embedded in) C or C++ source code. These SQL statements are translated by a SQL preprocessor into C or C++ source code, which you then compile.

Embedded SQL

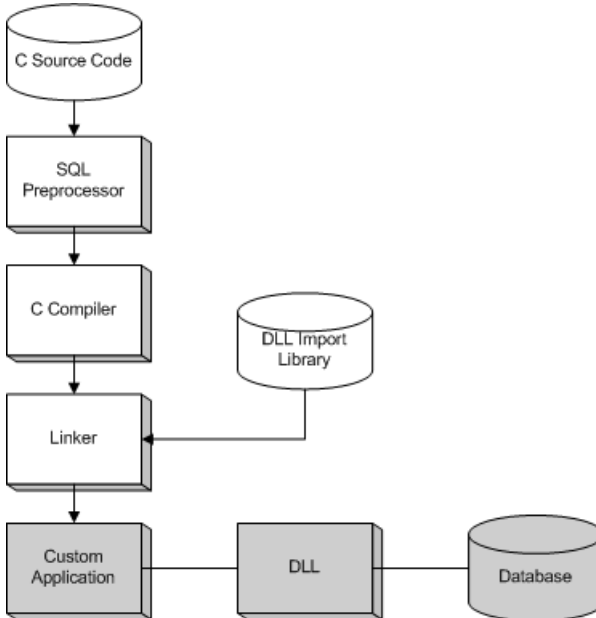
At runtime, embedded SQL applications use an SAP Sybase IQ *interface library* called DBLIB to communicate with a database server. DBLIB is a dynamic link library (DLL) or shared object on most platforms.

- On Windows operating systems, the interface library is `dblib16.dll`.
- On Unix operating systems, the interface library is `libdblib16.so`, `libdblib16.sl`, or `libdblib16.a`, depending on the operating system.
- On Mac OS X, the interface library is `libdblib16.dylib.1`.

SAP Sybase IQ provides two flavors of embedded SQL. Static embedded SQL is simpler to use, but is less flexible than dynamic embedded SQL.

Development Process Overview

Overview of the embedded SQL development process.



Once the program has been successfully preprocessed and compiled, it is linked with the *import library* for DBLIB to form an executable file. When the database server is running, this executable file uses DBLIB to interact with the database server. The database server does not have to be running when the program is preprocessed.

For Windows, there are 32-bit and 64-bit import libraries for Microsoft Visual C++. The use of import libraries is one method for developing applications that call functions in DLLs. However, it is recommended that the library be dynamically loaded, avoiding the use of import libraries.

The SQL Preprocessor

The SQL preprocessor is an executable named `iqiqlpp`.

The preprocessor command line is as follows:

```
iqiqlpp [ options ] sql-filename [ output-filename ]
```

The preprocessor translates the embedded SQL statements in a C or C++ source file into C code and places the result in an output file. A C or C++ compiler is then used to process the output file. The normal extension for source programs with embedded SQL is `.sqlc`. The default output file name is the *sql-filename* with an extension of `.c`. If the *sql-filename* has a `.c` extension, then the default output file name extension will change to `.cc`.

Note: When an application is rebuilt to use a new major version of the database interface library, the embedded SQL files must be preprocessed with the same version's SQL preprocessor.

The following table describes the preprocessor options.

Option	Description
<code>-d</code>	Generate code that reduces data space size. Data structures are reused and initialized at execution time before use. This increases code size.

Option	Description
<p><code>-e <i>level</i></code></p>	<p>Flag as an error any static embedded SQL that is not part of a specified standard. The <i>level</i> value indicates the standard to use. For example, <code>iqsqlpp -e c03 . . .</code> flags any syntax that is not part of the core SQL/2008 standard. The supported <i>level</i> values are:</p> <ul style="list-style-type: none"> • c08 – Flag syntax that is not core SQL/2008 syntax • p08 – Flag syntax that is not full SQL/2008 syntax • c03 – Flag syntax that is not core SQL/2003 syntax • p03 – Flag syntax that is not full SQL/2003 syntax • c99 – Flag syntax that is not core SQL/1999 syntax • p99 – Flag syntax that is not full SQL/1999 syntax • e92 – Flag syntax that is not entry-level SQL/1992 syntax • i92 – Flag syntax that is not intermediate-level SQL/1992 syntax • f92 – Flag syntax that is not full-SQL/1992 syntax • t – Flag non-standard host variable types • u – Flag syntax that is not supported by UltraLite <p>For compatibility with previous SAP Sybase IQ versions, you can also specify <i>e</i>, <i>i</i>, and <i>f</i>, which correspond to <i>e92</i>, <i>i92</i>, and <i>f92</i>, respectively.</p>
<p><code>-h <i>width</i></code></p>	<p>Limit the maximum length of lines output by <code>iqsqlpp</code> to <i>width</i>. The continuation character is a backslash (\) and the minimum value of <i>width</i> is 10.</p>
<p><code>-k</code></p>	<p>Notify the preprocessor that the program to be compiled includes a user declaration of <code>SQLCODE</code>. The definition must be of type <code>long</code>, but does not need to be in a declaration section.</p>

Option	Description
-m <i>mode</i>	<p>Specify the cursor updatability mode if it is not specified explicitly in the embedded SQL application. The <i>mode</i> can be one of:</p> <ul style="list-style-type: none"> • HISTORICAL – In previous versions, embedded SQL cursors defaulted to either FOR UPDATE or READ ONLY (depending on the query and the <code>ansi_update_constraints</code> option value). Explicit cursor updatability was specified on DECLARE CURSOR. Use this option to preserve this behavior. • READONLY – Embedded SQL cursors default to READ ONLY. Explicit cursor updatability is specified on PREPARE. This is the default behavior. READ ONLY cursors can result in improved performance.
-n	<p>Generate line number information in the C file. This consists of <code>#line</code> directives in the appropriate places in the generated C code. If the compiler that you are using supports the <code>#line</code> directive, this option makes the compiler report errors on line numbers in the SQC file (the one with the embedded SQL) as opposed to reporting errors on line numbers in the C file generated by the SQL preprocessor. Also, the <code>#line</code> directives are used indirectly by the source level debugger so that you can debug while viewing the SQC source file.</p>
-o <i>operating-system</i>	<p>Specify the target operating system. The supported operating systems are:</p> <ul style="list-style-type: none"> • WINDOWS – Microsoft Windows. • UNIX – Use this option if you are creating a 32-bit Unix application. • UNIX64 – Use this option if you are creating a 64-bit Unix application.
-q	Quiet mode—do not print messages.
-r-	Generate non-reentrant code.

Option	Description
<p><code>-s len</code></p>	<p>Set the maximum size string that the preprocessor puts into the C file. Strings longer than this value are initialized using a list of characters (<code>'a', 'b', 'c',</code> and so on). Most C compilers have a limit on the size of string literal they can handle. This option is used to set that upper limit. The default value is 500.</p>
<p><code>-u</code></p>	<p>Generate code for UltraLite.</p>
<p><code>-w level</code></p>	<p>Flag as a warning any static embedded SQL that is not part of a specified standard. The <i>level</i> value indicates the standard to use. For example, <code>iqsqlpp -w c08 . . .</code> flags any SQL syntax that is not part of the core SQL/2008 syntax. The supported <i>level</i> values are:</p> <ul style="list-style-type: none"> • c08 – Flag syntax that is not core SQL/2008 syntax • p08 – Flag syntax that is not full SQL/2008 syntax • c03 – Flag syntax that is not core SQL/2003 syntax • p03 – Flag syntax that is not full SQL/2003 syntax • c99 – Flag syntax that is not core SQL/1999 syntax • p99 – Flag syntax that is not full SQL/1999 syntax • e92 – Flag syntax that is not entry-level SQL/1992 syntax • i92 – Flag syntax that is not intermediate-level SQL/1992 syntax • f92 – Flag syntax that is not full-SQL/1992 syntax • t – Flag non-standard host variable types • u – Flag syntax that is not supported by UltraLite <p>For compatibility with previous SAP Sybase IQ versions, you can also specify <code>e</code>, <code>i</code>, and <code>f</code>, which correspond to <code>e92</code>, <code>i92</code>, and <code>f92</code>, respectively.</p>

Option	Description
<code>-x</code>	Change multibyte strings to escape sequences so that they can pass through compilers.
<code>-z cs</code>	Specify the collation sequence. For a list of recommended collation sequences, run <code>iqinit -l</code> at a command prompt. The collation sequence is used to help the preprocessor understand the characters used in the source code of the program, for example, in identifying alphabetic characters suitable for use in identifiers. If <code>-z</code> is not specified, the preprocessor attempts to determine a reasonable collation to use based on the operating system and the <code>SAL-ANG</code> and <code>SACHARSET</code> environment variables.
<i>sql-filename</i>	A C or C++ program containing embedded SQL to be processed.
<i>output-filename</i>	The C language source file created by the SQL preprocessor.

Supported Compilers

The C language SQL preprocessor has been used with the following compilers:

Operating system	Compiler	Version
Windows	Microsoft Visual C++	6.0 or later
Windows Mobile	Microsoft Visual C++	2005
Unix	GNU or native compiler	

Embedded SQL Header Files

All header files are installed in the `SDK\Include` subdirectory of your SAP Sybase IQ installation directory.

File name	Description
<code>sqlca.h</code>	Main header file included in all embedded SQL programs. This file includes the structure definition for the SQL Communication Area (SQLCA) and prototypes for all embedded SQL database interface functions.
<code>sqlda.h</code>	SQL Descriptor Area structure definition included in embedded SQL programs that use dynamic SQL.
<code>sqldef.h</code>	Definition of embedded SQL interface data types. This file also contains structure definitions and return codes needed for starting the database server from a C program.
<code>sqlerr.h</code>	Definitions for error codes returned in the <code>sqlcode</code> field of the SQLCA.
<code>sqlstate.h</code>	Definitions for ANSI/ISO SQL standard error states returned in the <code>sqlstate</code> field of the SQLCA.
<code>pshpk1.h</code> , <code>pshpk4.h</code> , <code>poppk.h</code>	These headers ensure that structure packing is handled correctly.

Import Libraries

On Windows platforms, all import libraries are installed in the `SDK\Lib` subdirectories, under the SAP Sybase IQ installation directory. Windows import libraries are stored in the `SDK\Lib\x86` and `SDK\Lib\x64` subdirectories. An export definition list is stored in `SDK\Lib\Def\dblib.def`.

On Unix platforms, all import libraries are installed in the `lib32` and `lib64` subdirectories, under the SAP Sybase IQ installation directory.

Operating system	Compiler	Import library
Windows	Microsoft Visual C++	<code>dblibtm.lib</code>

Operating system	Compiler	Import library
Unix (unthreaded applications)	All compilers	libdblib16.so, libdbtasks16.so, libdblib16.sl, libdbtasks16.sl
Unix (threaded applications)	All compilers	libdblib16_r.so, libdb- tasks16_r.so,libd- blib16_r.sl,libdb- tasks16_r.sl

The libdbtasks16 libraries are called by the libdblib16 libraries. Some compilers locate libdbtasks16 automatically. For others, you need to specify it explicitly.

Sample Embedded SQL Program

The following is a very simple example of an embedded SQL program.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main()
{
    db_init( &sqlca );
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    EXEC SQL UPDATE Employees
        SET Surname = 'Plankton'
        WHERE EmployeeID = 195;
    EXEC SQL COMMIT WORK;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( 0 );
error:
    printf( "update unsuccessful -- sqlcode = %ld\n",
        sqlca.sqlcode );
    db_fini( &sqlca );
    return( -1 );
}
```

This example connects to the database, updates the last name of employee number 195, commits the change, and exits. There is virtually no interaction between the embedded SQL code and the C code. The only thing the C code is used for in this example is control flow. The WHENEVER statement is used for error checking. The error action (GOTO in this example) is executed after any SQL statement that causes an error.

Structure of Embedded SQL Programs

SQL statements are placed (embedded) within regular C or C++ code. All embedded SQL statements start with the words EXEC SQL and end with a semicolon (;). Normal C language comments are allowed in the middle of embedded SQL statements.

Every C program using embedded SQL must contain the following statement before any other embedded SQL statements in the source file.

```
EXEC SQL INCLUDE SQLCA;
```

Every C program using embedded SQL must initialize a SQLCA first:

```
db_init( &sqlca );
```

One of the first embedded SQL statements executed by the C program must be a CONNECT statement. The CONNECT statement is used to establish a connection with the database server and to specify the user ID that is used for authorizing all statements executed during the connection.

Some embedded SQL statements do not generate any C code, or do not involve communication with the database. These statements are allowed before the CONNECT statement. Most notable are the INCLUDE statement and the WHENEVER statement for specifying error processing.

Every C program using embedded SQL must finalize any SQLCA that has been initialized.

```
db_fini( &sqlca );
```

Loading DBLIB Dynamically Under Windows

Load DBLIB dynamically from your embedded SQL application using the `esqldll.c` module in the `SDK\C` subdirectory of your installation directory so that you do not need to link against the import library.

Prerequisites

There are no prerequisites for this task.

Task

This task is an alternative to the usual technique of linking an application against a static import library for a Dynamic Link Library (DLL) that contains the required function definitions.

A similar task can be used to dynamically load DBLIB on Unix platforms.

1. Your application must call `db_init_dll` to load the DBLIB DLL, and must call `db_fini_dll` to free the DBLIB DLL. The `db_init_dll` call must be before any function in the database interface, and no function in the interface can be called after `db_fini_dll`.

You must still call the `db_init` and `db_fini` library functions.

2. You must include the `esqldll.h` header file before the EXEC SQL INCLUDE SQLCA statement or include `sqlca.h` in your embedded SQL program. The `esqldll.h` header file includes `sqlca.h`.
3. A SQL OS macro must be defined. The header file `sqlos.h`, which is included by `sqlca.h`, attempts to determine the appropriate macro and define it. However, certain combinations of platforms and compilers may cause this to fail. In this case, you must add a `#define` to the top of this file, or make the definition using a compiler option. The macro that must be defined for Windows is shown below.

Macro	Platforms
<code>_SQL_OS_WINDOWS</code>	All Windows operating systems

4. Compile `esqldll.c`.
5. Instead of linking against the import library, link the object module `esqldll.obj` with your embedded SQL application objects.

The DBLIB interface DLL loads dynamically when you run your embedded SQL application.

Sample Embedded SQL Programs

Sample embedded SQL programs are included with the SAP Sybase IQ installation. They are placed in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C` directory.

- The static cursor embedded SQL example, `cur.sqc`, demonstrates the use of static SQL statements.
- The dynamic cursor embedded SQL example, `dcur.sqc`, demonstrates the use of dynamic SQL statements.

To reduce the amount of code that is duplicated by the sample programs, the mainlines and the data printing functions have been placed into a separate file. This is `mainch.c` for character mode systems and `mainwin.c` for windowing environments.

The sample programs each supply the following three routines, which are called from the mainlines:

- **WSQLEX_Init** – Connects to the database and opens the cursor.
- **WSQLEX_Process_Command** – Processes commands from the user, manipulating the cursor as necessary.

- **WSQLEX_Finish** – Closes the cursor and disconnects from the database.

The function of the mainline is to:

1. Call the **WSQLEX_Init** routine.
2. Loop, getting commands from the user and calling **WSQL_Process_Command** until the user quits.
3. Call the **WSQLEX_Finish** routine.

Connecting to the database is done with the embedded SQL **CONNECT** statement supplying the appropriate user ID and password.

In addition to these samples, you may find other programs and source files as part of SAP Sybase IQ that demonstrate features available for particular platforms.

Static Cursor Sample

This example demonstrates the use of cursors. The particular cursor used here retrieves certain information from the **Employees** table in the sample database. The cursor is declared statically, meaning that the actual SQL statement to retrieve the information is hard coded into the source program. This is a good starting point for learning how cursors work. The **Dynamic Cursor** sample takes this first example and converts it to use dynamic SQL statements.

The **open_cursor** routine both declares a cursor for the specific SQL query and also opens the cursor.

Printing a page of information is done by the **print** routine. It loops *pagesize* times, fetching a single row from the cursor and printing it out. The **fetch** routine checks for warning conditions, such as rows that cannot be found (**SQLCODE 100**), and prints appropriate messages when they arise. In addition, the cursor is repositioned by this program to the row before the one that appears at the top of the current page of data.

The **move**, **top**, and **bottom** routines use the appropriate form of the **FETCH** statement to position the cursor. This form of the **FETCH** statement doesn't actually get the data—it only positions the cursor. Also, a general relative positioning routine, **move**, has been implemented to move in either direction depending on the sign of the parameter.

When the user quits, the cursor is closed and the database connection is also released. The cursor is closed by a **ROLLBACK WORK** statement, and the connection is released by a **DISCONNECT**.

Running the Static Cursor Sample Program

Run the static cursor sample program.

Prerequisites

There are no prerequisites for this task.

Task

The executable files and corresponding source code are located in the %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C directory.

1. Start the SAP Sybase IQ sample database, `iqdemo.db`.
2. Files to build the sample programs are supplied with the sample code.

To build the 32-bit samples on Windows, use `build.bat`.

To build the 64-bit samples on Windows, use `build64.bat`. You may need to set up the correct environment for compiling and linking. Here is an example that builds the sample programs for an x64 platform.

```
set mssdk=c:\MSSDK\v7.0
build64
```

To build the samples on Unix, use the shell script `build.sh`.

3. For the 32-bit Windows example, run the file `curwin.exe`.

For the 64-bit Windows example, run the file `curx64.exe`.

For the Unix example, run the file `cur`.

4. Follow the on-screen instructions.

The various commands manipulate a database cursor and print the query results on the screen. Enter the letter of the command that you want to perform. Some systems may require you to press Enter after the letter.

Dynamic Cursor Sample

This sample demonstrates the use of cursors for a dynamic SQL SELECT statement.

The dynamic cursor sample program (`dcur`) allows the user to select a table to look at with the `N` command. The program then presents as much information from that table as fits on the screen.

When this program is run, it prompts for a connection string. The following is an example.

```
UID=<userid>;PWD=<your_password>;DBF=iqdemo.db
```

The C program with the embedded SQL is located in the %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C directory.

The `dcur` program uses the embedded SQL interface function `db_string_connect` to connect to the database. This function provides the extra functionality to support the connection string that is used to connect to the database.

The `open_cursor` routine first builds the SELECT statement

```
SELECT * FROM table-name
```

where *table-name* is a parameter passed to the routine. It then prepares a dynamic SQL statement using this string.

The embedded SQL DESCRIBE statement is used to fill in the SQLDA structure with the results of the SELECT statement.

Note: An initial guess is taken for the size of the SQLDA (3). If this is not big enough, the actual size of the SELECT list returned by the database server is used to allocate a SQLDA of the correct size.

The SQLDA structure is then filled with buffers to hold strings that represent the results of the query. The fill_s_sqlda routine converts all data types in the SQLDA to DT_STRING and allocates buffers of the appropriate size.

A cursor is then declared and opened for this statement. The rest of the routines for moving and closing the cursor remain the same.

The fetch routine is slightly different: it puts the results into the SQLDA structure instead of into a list of host variables. The print routine has changed significantly to print results from the SQLDA structure up to the width of the screen. The print routine also uses the name fields of the SQLDA to print headings for each column.

Running the Dynamic Cursor Sample Program

Run the dynamic cursor sample program.

Prerequisites

There are no prerequisites for this task.

Task

The executable files and corresponding source code are located in the %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C directory. For Windows Mobile, an additional example is located in the \SQLAnywhere\CE\esql_sample directory.

1. Start the SAP Sybase IQ sample database, iqdemo.db.
2. Files to build the sample programs are supplied with the sample code.

To build the 32-bit samples on Windows, use build.bat.

To build the 64-bit samples on Windows, use build64.bat. You may need to set up the correct environment for compiling and linking. Here is an example that builds the sample programs for an x64 platform.

```
set mssdk=c:\MSSDK\v7.0  
build64
```

For Windows Mobile, use the esql_sample.sln project file for Microsoft Visual C ++. This file appears in SQLAnywhere\CE\esql_sample.

To build the samples on Unix, use the shell script `build.sh`.

- For the 32-bit Windows example, run the file `dcurwin.exe`.

For the 64-bit Windows example, run the file `dcurx64.exe`.

For the Windows Mobile example, deploy and run the file `esql_sample.exe` on your Windows Mobile device.

For the Unix example, run the file `dcur`.

- Each sample program presents a console-type user interface and prompts you for a command. Enter the following connection string to connect to the sample database:

```
DSN=Sybase IQ Demo
```

- Each sample program prompts you for a table. Choose one of the tables in the sample database. For example, you can enter `Customers` or `Employees`.
- Follow the on-screen instructions.

The various commands manipulate a database cursor and print the query results on the screen. Enter the letter of the command that you want to perform. Some systems may require you to press Enter after the letter.

Embedded SQL Data Types

To transfer information between a program and the database server, every piece of data must have a data type. The embedded SQL data type constants are prefixed with `DT_`, and can be found in the `sqldef.h` header file. You can create a host variable of any one of the supported types. You can also use these types in a `SQLDA` structure for passing data to and from the database.

You can define variables of these data types using the `DECL_` macros listed in `sqlca.h`. For example, a variable holding a `BIGINT` value could be declared with `DECL_BIGINT`.

The following data types are supported by the embedded SQL programming interface:

- `DT_BIT` – 8-bit signed integer.
- `DT_SMALLINT` – 16-bit signed integer.
- `DT_UNSSMALLINT` – 16-bit unsigned integer.
- `DT_TINYINT` – 8-bit signed integer.
- `DT_BIGINT` – 64-bit signed integer.
- `DT_UNSBIGINT` – 64-bit unsigned integer.
- `DT_INT` – 32-bit signed integer.
- `DT_UNSENT` – 32-bit unsigned integer.
- `DT_FLOAT` – 4-byte floating-point number.
- `DT_DOUBLE` – 8-byte floating-point number.

- **DT_DECIMAL** – Packed decimal number (proprietary format).

```
typedef struct TYPE_DECIMAL {
    char array[1];
} TYPE_DECIMAL;
```

- **DT_STRING** – Null-terminated character string, in the CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.
- **DT_NSTRING** – Null-terminated character string, in the NCHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.
- **DT_DATE** – Null-terminated character string that is a valid date.
- **DT_TIME** – Null-terminated character string that is a valid time.
- **DT_TIMESTAMP** – Null-terminated character string that is a valid timestamp.
- **DT_FIXCHAR** – Fixed-length blank-padded character string, in the CHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.
- **DT_NFIXCHAR** – Fixed-length blank-padded character string, in the NCHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.
- **DT_VARCHAR** – Varying length character string, in the CHAR character set, with a two-byte length field. The maximum length is 32765 bytes. When sending data, you must set the length field. When fetching data, the database server sets the length field. The data is not null-terminated or blank-padded.

```
typedef struct VARCHAR {
    a_sql_ulen len;
    char array[1];
} VARCHAR;
```

- **DT_NVARCHAR** – Varying length character string, in the NCHAR character set, with a two-byte length field. The maximum length is 32765 bytes. When sending data, you must set the length field. When fetching data, the database server sets the length field. The data is not null-terminated or blank-padded.

```
typedef struct NVARCHAR {
    a_sql_ulen len;
    char array[1];
} NVARCHAR;
```

- **DT_LONGVARCHAR** – Long varying length character string, in the CHAR character set.

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
    * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated
    expression
    * (may be larger than array_len) */
    char array[1]; /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

The LONGVARCHAR structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can

be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null-terminated or blank-padded.

- **DT_LONGNVARCHAR** – Long varying length character string, in the NCHAR character set. The macro defines a structure, as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
                             * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated
                             * expression
                             * (may be larger than array_len) */
    char          array[1]; /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

The LONGNVARCHAR structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null-terminated or blank-padded.

- **DT_BINARY** – Varying length binary data with a two-byte length field. The maximum length is 32765 bytes. When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field.

```
typedef struct BINARY {
    a_sql_ulen len;
    char          array[1];
} BINARY;
```

- **DT_LONGBINARY** – Long binary data. The macro defines a structure, as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
                             * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated
                             * expression
                             * (may be larger than array_len) */
    char          array[1]; /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

The LONGBINARY structure may be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement.

- **DT_TIMESTAMP_STRUCT** – SQLDATETIME structure with fields for each part of a timestamp.

```
typedef struct sqldatetime {
    unsigned short year; /* for example 1999 */
    unsigned char  month; /* 0-11 */
    unsigned char  day_of_week; /* 0-6 0=Sunday */
    unsigned short day_of_year; /* 0-365 */
    unsigned char  day; /* 1-31 */
    unsigned char  hour; /* 0-23 */
}
```

Embedded SQL

```
unsigned char minute; /* 0-59 */
unsigned char second; /* 0-59 */
unsigned long microsecond; /* 0-999999 */
} SQLDATETIME;
```

The `SQLDATETIME` structure can be used to retrieve fields of `DATE`, `TIME`, and `TIMESTAMP` type (or anything that can be converted to one of these). Often, applications have their own formats and date manipulation code. Fetching data in this structure makes it easier for you to manipulate this data. `DATE`, `TIME`, and `TIMESTAMP` fields can also be fetched and updated with any character type.

If you use a `SQLDATETIME` structure to enter a date, time, or timestamp into the database, the `day_of_year` and `day_of_week` members are ignored.

- **DT_VARIABLE** – Null-terminated character string. The character string must be the name of a SQL variable whose value is used by the database server. This data type is used only for supplying data to the database server. It cannot be used when fetching data from the database server.

The structures are defined in the `sqlca.h` file. The `VARCHAR`, `NVARCHAR`, `BINARY`, `DECIMAL`, and `LONG` data types are not useful for declaring host variables because they contain a one-character array. However, they are useful for allocating variables dynamically or typecasting other variables.

DATE and TIME database types

There are no corresponding embedded SQL interface data types for the various `DATE` and `TIME` database types. These database types are all fetched and updated using either the `SQLDATETIME` structure or character strings.

Host Variables in Embedded SQL

Host variables are C variables that are identified to the SQL preprocessor. Host variables can be used to send values to the database server or receive values from the database server.

Host variables are quite easy to use, but they have some restrictions. Dynamic SQL is a more general way of passing information to and from the database server using a structure known as the SQL Descriptor Area (SQLDA). The SQL preprocessor automatically generates a SQLDA for each statement in which host variables are used.

Host variables cannot be used in batches. Host variables cannot be used within a subquery in a `SET` statement.

Host Variable Declaration

Host variables are defined by putting them into a declaration section. According to the ANSI embedded SQL standard, host variables are defined by surrounding the normal C variable declarations with the following:

```
EXEC SQL BEGIN DECLARE SECTION;
/* C variable declarations */
EXEC SQL END DECLARE SECTION;
```

These host variables can then be used in place of value constants in any SQL statement. When the database server executes the statement, the value of the host variable is used. Host variables cannot be used in place of table or column names: dynamic SQL is required for this. The variable name is prefixed with a colon (:) in a SQL statement to distinguish it from other identifiers allowed in the statement.

In the SQL preprocessor, C language code is only scanned inside a DECLARE SECTION. So, TYPEDEF types and structures are not allowed, but initializers on the variables are allowed inside a DECLARE SECTION.

Example

The following sample code illustrates the use of host variables on an INSERT statement. The variables are filled in by the program and then inserted into the database:

```
EXEC SQL BEGIN DECLARE SECTION;
long employee_number;
char employee_name[50];
char employee_initials[8];
char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* program fills in variables with appropriate values
*/
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone );
```

C Host Variable Types

Only a limited number of C data types are supported as host variables. Also, certain host variable types do not have a corresponding C type.

Macros defined in the `sqlca.h` header file can be used to declare host variables of the following types: NCHAR, VARCHAR, NVARCHAR, LONGVARCHAR, LONGNVARCHAR, BINARY, LONGBINARY, DECIMAL, DT_FIXCHAR, DT_NFIXCHAR, DATETIME (SQLDATETIME), BIT, BIGINT, or UNSIGNED BIGINT. They are used as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_NCHAR          v_nchar[10];
DECL_VARCHAR( 10 ) v_varchar;
DECL_NVARCHAR( 10 ) v_nvarchar;
DECL_LONGVARCHAR( 32768 ) v_longvarchar;
```

Embedded SQL

```

DECL_LONGNVARCHAR( 32768 ) v_longnvarchar;
DECL_BINARY( 4000 ) v_binary;
DECL_LONGBINARY( 128000 ) v_longbinary;
DECL_DECIMAL( 30, 6 ) v_decimal;
DECL_FIXCHAR( 10 ) v_fixchar;
DECL_NFIXCHAR( 10 ) v_nfixchar;
DECL_DATETIME v_datetime;
DECL_BIT v_bit;
DECL_BIGINT v_bigint;
DECL_UNSIGNED_BIGINT v_ubigint;
EXEC SQL END DECLARE SECTION;

```

The preprocessor recognizes these macros within an embedded SQL declaration section and treats the variable as the appropriate type. It is recommended that the DECIMAL (DT_DECIMAL, DECL_DECIMAL) type not be used since the format of decimal numbers is proprietary.

The following table lists the C variable types that are allowed for host variables and their corresponding embedded SQL interface data types.

C data type	Embedded SQL interface type	Description
short si; short int si;	DT_SMALLINT	16-bit signed integer.
unsigned short int usi; usi;	DT_UNSSMALLINT	16-bit unsigned integer.
long l; long int l;	DT_INT	32-bit signed integer.
unsigned long int ul; ul;	DT_UNSENT	32-bit unsigned integer.
DECL_BIGINT ll;	DT_BIGINT	64-bit signed integer.
DECL_UNSIGNED_BIGINT ull; ull;	DT_UNSBIGINT	64-bit unsigned integer.
float f;	DT_FLOAT	4-byte single-precision floating-point value.
double d;	DT_DOUBLE	8-byte double-precision floating-point value.
char a[n]; /*n>=1*/	DT_STRING	Null-terminated string, in CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings. This variable holds n-1 bytes plus the null terminator.

C data type	Embedded SQL interface type	Description
<code>char *a;</code>	DT_STRING	Null-terminated string, in CHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.
<code>DECL_NCHAR a[n]; /*n>=1*/</code>	DT_NSTRING	Null-terminated string, in NCHAR character set. The string is blank-padded if the database is initialized with blank-padded strings. This variable holds n-1 bytes plus the null terminator.
<code>DECL_NCHAR *a;</code>	DT_NSTRING	Null-terminated string, in NCHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.
<code>DECL_VARCHAR(n) a;</code>	DT_VARCHAR	Varying length character string, in CHAR character set, with 2-byte length field. Not null-terminated or blank-padded. The maximum value for n is 32765 (bytes).
<code>DECL_NVARCHAR(n) a;</code>	DT_NVARCHAR	Varying length character string, in NCHAR character set, with 2-byte length field. Not null-terminated or blank-padded. The maximum value for n is 32765 (bytes).
<code>DECL_LONGVARCHAR(n) a;</code>	DT_LONGVARCHAR	Varying length long character string, in CHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.

C data type	Embedded SQL interface type	Description
DECL_LONGNVARCHAR (n) a;	DT_LONGNVARCHAR	Varying length long character string, in NCHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.
DECL_BINARY (n) a;	DT_BINARY	Varying length binary data with 2-byte length field. The maximum value for n is 32765 (bytes).
DECL_LONGBINARY (n) a;	DT_LONGBINARY	Varying length long binary data with three 4-byte length fields.
char a; / *n=1*/ DECL_FIXCHAR (n) a;	DT_FIXCHAR	Fixed length character string, in CHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).
DECL_NCHAR a; / *n=1*/ DECL_NFIXCHAR (n) a;	DT_NFIXCHAR	Fixed length character string, in NCHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).
DECL_DATETIME a;	DT_TIMESTAMP_STRUCT	SQLDATETIME structure

Character sets

For DT_FIXCHAR, DT_STRING, DT_VARCHAR, and DT_LONGVARCHAR, character data is in the application's CHAR character set, which is usually the character set of the application's locale. An application can change the CHAR character set either by using the CHARSET connection parameter, or by calling the db_change_char_charset function.

For DT_NFIXCHAR, DT_NSTRING, DT_NVARCHAR, and DT_LONGNVARCHAR, data is in the application's NCHAR character set. By default, the application's NCHAR character set is the same as the CHAR character set. An application can change the NCHAR character set by calling the db_change_nchar_charset function.

Data lengths

Regardless of the CHAR and NCHAR character sets in use, all data lengths are specified in bytes.

If character set conversion occurs between the server and the application, it is the application's responsibility to ensure that buffers are sufficiently large to handle the converted data, and to issue additional GET DATA statements if data is truncated.

Pointers to char

The database interface considers a host variable declared as a *pointer to char* (`char * a`) to be 32767 bytes long. Any host variable of type pointer to char used to retrieve information from the database must point to a buffer large enough to hold any value that could possibly come back from the database.

This is potentially quite dangerous because someone could change the definition of the column in the database to be larger than it was when the program was written. This could cause random memory corruption problems. It is better to use a declared array, even as a parameter to a function, where it is passed as a pointer to char. This technique allows the embedded SQL statements to know the size of the array.

Scope of host variables

A standard host-variable declaration section can appear anywhere that C variables can normally be declared. This includes the parameter declaration section of a C function. The C variables have their normal scope (available within the block in which they are defined). However, since the SQL preprocessor does not scan C code, it does not respect C blocks.

As far as the SQL preprocessor is concerned, host variables are global to the source file; two host variables cannot have the same name.

Host Variable Usage

Host variables can be used in the following circumstances:

- SELECT, INSERT, UPDATE, and DELETE statements in any place where a number or string constant is allowed.
- The INTO clause of SELECT and FETCH statements.
- Host variables can also be used in place of a statement name, a cursor name, or an option name in statements specific to embedded SQL.
- For CONNECT, DISCONNECT, and SET CONNECT statements, a host variable can be used in place of a server name, database name, connection name, user ID, password, or connection string.
- For SET OPTION and GET OPTION, a host variable can be used in place of the option value.

Host variables cannot be used in the following circumstances:

- Host variables cannot be used in place of a table name or a column name in any statement.
- Host variables cannot be used in batches.
- Host variables cannot be used within a subquery in a SET statement.

SQLCODE and SQLSTATE host variables

The ISO/ANSI standard allows an embedded SQL source file to declare the following special host variables within an embedded SQL declaration section:

```
long SQLCODE;  
char SQLSTATE[6];
```

If used, these variables are set after any embedded SQL statement that makes a database request (EXEC SQL statements other than DECLARE SECTION, INCLUDE, WHENEVER SQLCODE, and so on). As a consequence, the SQLCODE and SQLSTATE host variables must be visible in the scope of every embedded SQL statement that generates database requests.

The following is valid embedded SQL:

```
EXEC SQL INCLUDE SQLCA;  
// declare SQLCODE with global scope  
EXEC SQL BEGIN DECLARE SECTION;  
long SQLCODE;  
EXEC SQL END DECLARE SECTION;  
sub1() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    char SQLSTATE[6];  
    EXEC SQL END DECLARE SECTION;  
    exec SQL CREATE TABLE ...  
}  
sub2() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    char SQLSTATE[6];  
    EXEC SQL END DECLARE SECTION;  
    exec SQL DROP TABLE ...  
}
```

The following is not valid embedded SQL because SQLSTATE is not defined in the scope of the function sub2:

```
EXEC SQL INCLUDE SQLCA;  
sub1() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    char SQLSTATE[6];  
    EXEC SQL END DECLARE SECTION;  
    exec SQL CREATE TABLE...  
}  
sub2() {  
    exec SQL DROP TABLE...  
}
```

Indicator Variables

Indicator variables are C variables that hold supplementary information when you are fetching or putting data. There are several distinct uses for indicator variables:

- **NULL values** – To enable applications to handle NULL values.

- **String truncation** – To enable applications to handle cases when fetched values must be truncated to fit into host variables.
- **Conversion errors** – To hold error information.

An indicator variable is a host variable of type `a_sql_len` that is placed immediately following a regular host variable in a SQL statement. For example, in the following INSERT statement, `:ind_phone` is an indicator variable:

```
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

On a fetch or execute where no rows are received from the database server (such as when an error or end of result set occurs), then indicator values are unchanged.

Note: To allow for the future use of 32 and 64-bit lengths and indicators, the use of short int for embedded SQL indicator variables is deprecated. Use `a_sql_len` instead.

Indicator Variables: The SQL NULL Value

Do not confuse the SQL concept of NULL with the C-language constant of the same name. In the SQL language, NULL represents either an unknown attribute or inapplicable information. The C-language constant represents a pointer value that does not point to a memory location.

When NULL is used in the SAP Sybase IQ documentation, it refers to the SQL database meaning given above. The C language constant is referred to as the null pointer (lowercase).

NULL is not the same as any value of the column's defined type. So, something extra is required beyond regular host variables to pass NULL values to the database or receive NULL results back. Indicator variables are used for this purpose.

Using indicator variables when inserting NULL

An INSERT statement could include an indicator variable as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/*
This program fills in the employee number,
name, initials, and phone number.
*/
if( /* Phone number is unknown */ ) {
    ind_phone = -1;
} else {
    ind_phone = 0;
}
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

If the indicator variable has a value of -1, a NULL is written. If it has a value of 0, the actual value of employee_phone is written.

Using indicator variables when fetching NULL

Indicator variables are also used when receiving data from the database. They are used to indicate that a NULL value was fetched (indicator is negative). If a NULL value is fetched from the database and an indicator variable is not supplied, an error is generated (SQLE_NO_INDICATOR).

Indicator Variables: Truncated Values

Indicator variables indicate whether any fetched values were truncated to fit into a host variable. This enables applications to handle truncation appropriately.

If a value is truncated on fetching, the indicator variable is set to a positive value, containing the actual length of the database value before truncation. If the actual length of the database value is greater than 32767 bytes, then the indicator variable contains 32767.

Indicator Variables: Conversion Errors

By default, the conversion_error database option is set to On, and any data type conversion failure leads to an error, with no row returned.

You can use indicator variables to tell which column produced a data type conversion failure. If you set the database option conversion_error to Off, any data type conversion failure gives a CANNOT_CONVERT warning, rather than an error. If the column that suffered the conversion error has an indicator variable, that variable is set to a value of -2.

If you set the conversion_error option to Off when inserting data into the database, a value of NULL is inserted when a conversion failure occurs.

Summary of Indicator Variable Values

The following table provides a summary of indicator variable usage.

Indicator value	Supplying value to database	Receiving value from database
> 0	Host variable value	Retrieved value was truncated —actual length in indicator variable.
0	Host variable value	Fetch successful, or conversion_error set to On.
-1	NULL value	NULL result.
-2	NULL value	Conversion error (when conversion_error is set to Off only). SQLCODE indicates a CANNOT_CONVERT warning.

Indicator value	Supplying value to database	Receiving value from database
< -2	NULL value	NULL result.

The SQL Communication Area (SQLCA)

The SQL Communication Area (SQLCA) is an area of memory that is used on every database request for communicating statistics and errors from the application to the database server and back to the application. The SQLCA is used as a handle for the application-to-database communication link. It is passed in to all database library functions that need to communicate with the database server. It is implicitly passed on all embedded SQL statements.

A global SQLCA variable is defined in the interface library. The preprocessor generates an external reference for the global SQLCA variable and an external reference for a pointer to it. The external reference is named `sqlca` and is of type `SQLCA`. The pointer is named `sqlcaptr`. The actual global variable is declared in the import library.

The SQLCA is defined by the `sqlca.h` header file, included in the `SDK\Include` subdirectory of your installation directory.

SQLCA provides error codes

You reference the SQLCA to test for a particular error code. The `sqlcode` and `sqlstate` fields contain error codes when a database request has an error. Some C macros are defined for referencing the `sqlcode` field, the `sqlstate` field, and some other fields.

SQLCA Fields

The fields in the SQLCA have the following meanings:

- **sqlcaid** – An 8-byte character field that contains the string `SQLCA` as an identification of the SQLCA structure. This field helps in debugging when you are looking at memory contents.
- **sqlcabc** – A 32-bit integer that contains the length of the SQLCA structure (136 bytes).
- **sqlcode** – A 32-bit integer that specifies the error code when the database detects an error on a request. Definitions for the error codes can be found in the header file `sqlerr.h`. The error code is 0 (zero) for a successful operation, positive for a warning, and negative for an error.
- **sqlerrml** – The length of the information in the `sqlerrmc` field.
- **sqlerrmc** – Zero or more character strings to be inserted into an error message. Some error messages contain one or more placeholder strings (`%1`, `%2`, ...) that are replaced with the strings in this field.

For example, if a `Table Not Found` error is generated, `sqlerrmc` contains the table name, which is inserted into the error message at the appropriate place.

- **sqlerrp** – Reserved.
- **sqlerrd** – A utility array of 32-bit integers.
- **sqlwarn** – Reserved.
- **sqlstate** – The SQLSTATE status value. The ANSI SQL standard defines this type of return value from a SQL statement in addition to the SQLCODE value. The SQLSTATE value is always a five-character null-terminated string, divided into a two-character class (the first two characters) and a three-character subclass. Each character can be a digit from 0 through 9 or an uppercase alphabetic character A through Z.

Any class or subclass that begins with 0 through 4 or A through H is defined by the SQL standard; other classes and subclasses are implementation defined. The SQLSTATE value '00000' means that there has been no error or warning.

sqlerror array

The sqlerror field array has the following elements.

- **sqlerrd[1] (SQLIOCOUNT)** – The actual number of input/output operations that were required to complete a statement.

The database server does not set this number to zero for each statement. Your program can set this variable to zero before executing a sequence of statements. After the last statement, this number is the total number of input/output operations for the entire statement sequence.

- **sqlerrd[2] (SQLCOUNT)** – The value of this field depends on which statement is being executed.
 - **INSERT, UPDATE, PUT, and DELETE statements** – The number of rows that were affected by the statement.
 - **OPEN and RESUME statements** – On a cursor OPEN or RESUME, this field is filled in with either the actual number of rows in the cursor (a value greater than or equal to 0) or an estimate thereof (a negative number whose absolute value is the estimate). It is the actual number of rows if the database server can compute it without counting the rows. The database can also be configured to always return the actual number of rows using the row_counts option.
 - **FETCH cursor statement** – The SQLCOUNT field is filled if a SQLE_NOTFOUND warning is returned. It contains the number of rows by which a FETCH RELATIVE or FETCH ABSOLUTE statement goes outside the range of possible cursor positions (a cursor can be on a row, before the first row, or after the last row). For a wide fetch, SQLCOUNT is the number of rows actually fetched, and is less than or equal to the number of rows requested. During a wide fetch, SQLE_NOTFOUND is only set if no rows are returned.

The value is 0 if the row was not found, but the position is valid, for example, executing FETCH RELATIVE 1 when positioned on the last row of a cursor. The value is positive

if the attempted fetch was beyond the end of the cursor, and negative if the attempted fetch was before the beginning of the cursor.

- **GET DATA statement** – The SQLCOUNT field holds the actual length of the value.
- **DESCRIBE statement** – If the WITH VARIABLE RESULT clause is used to describe procedures that may have more than one result set, SQLCOUNT is set to one of the following values:
 - **0** – The result set may change: the procedure call should be described again following each OPEN statement.
 - **1** – The result set is fixed. No re-describing is required.

For the SQLE_SYNTAX_ERROR syntax error, the field contains the approximate character position within the statement where the error was detected.

- **sqlerrd[3] (SQLIOESTIMATE)** – The estimated number of input/output operations that are required to complete the statement. This field is given a value on an OPEN or EXPLAIN statement.

SQLCA Management for Multithreaded or Reentrant Code

You can use embedded SQL statements in multithreaded or reentrant code. However, if you use a single connection, you are restricted to one active request per connection. In a multithreaded application, you should not use the same connection to the database on each thread unless you use a semaphore to control access.

There are no restrictions on using separate connections on each thread that wants to use the database. The SQLCA is used by the runtime library to distinguish between the different thread contexts. So, each thread wanting to use the database concurrently must have its own SQLCA. The exception is that a thread can use the `db_cancel_request` function to cancel a statement executing on a different thread using that thread's SQLCA.

The following is an example of reentrant multithreaded embedded SQL code.

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <process.h>
#include <windows.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

#define TRUE 1
#define FALSE 0

// multithreading support

typedef struct a_thread_data {
    SQLCA sqlca;
    int num_iters;
    int thread;
    int done;
```

```

} a_thread_data;

// each thread's ESQL test

EXEC SQL SET SQLCA "&thread_data->sqlca";

static void PrintSQLError( a_thread_data * thread_data )
/*****
{
    char                buffer[200];

    printf( "%d: SQL error %d -- %s ... aborting\n",
            thread_data->thread,
            SQLCODE,
            sqlerror_message( &thread_data->sqlca,
                             buffer, sizeof( buffer ) ) );
    exit( 1 );
}

EXEC SQL WHENEVER SQLERROR { PrintSQLError( thread_data ); };

static void do_one_iter( void * data )
{
    a_thread_data * thread_data = (a_thread_data *)data;
    int            i;
    EXEC SQL BEGIN DECLARE SECTION;
    char           user[ 20 ];
    EXEC SQL END DECLARE SECTION;

    if( db_init( &thread_data->sqlca ) != 0 ) {
        for( i = 0; i < thread_data->num_iters; i++ ) {
            EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";
            EXEC SQL SELECT USER INTO :user;
            EXEC SQL DISCONNECT;
        }
        printf( "Thread %d did %d iters successfully\n",
                thread_data->thread, thread_data->num_iters );
        db_fini( &thread_data->sqlca );
    }
    thread_data->done = TRUE;
}

int main()
{
    int num_threads = 4;
    int thread;
    int num_iters = 300;
    int num_done = 0;
    a_thread_data *thread_data;
    thread_data = (a_thread_data *)malloc( sizeof(a_thread_data) *
num_threads );
    for( thread = 0; thread < num_threads; thread++ ) {
        thread_data[ thread ].num_iters = num_iters;
        thread_data[ thread ].thread = thread;
        thread_data[ thread ].done = FALSE;
        if( _beginthread( do_one_iter,

```



```

    8096,
    (void *)&thread_data[thread] ) <= 0 ) {
    printf( "FAILED creating thread.\n" );
    return( 1 );
    }
}
while( num_done != num_threads ) {
    Sleep( 1000 );
    num_done = 0;
    for( thread = 0; thread < num_threads; thread++ ) {
        if( thread_data[ thread ].done == TRUE ) {
            num_done++;
        }
    }
}
return( 0 );
}

```

Multiple SQLCAs

You must not use the SQL preprocessor option (-r-) that generates non-reentrant code. Reentrant code is a little larger and a little slower because statically initialized global variables cannot be used. However, these effects are minimal.

Each SQLCA used in your program must be initialized with a call to `db_init` and cleaned up at the end with a call to `db_fini`.

The embedded SQL statement `SET SQLCA` is used to tell the SQL preprocessor to use a different SQLCA for database requests. Usually, a statement such as `EXEC SQL SET SQLCA 'task_data->sqlca'`; is used at the top of your program or in a header file to set the SQLCA reference to point at task specific data. Performance is unaffected because this statement does not generate any code. It changes the state within the preprocessor so that any reference to the SQLCA uses the given string.

Each thread must have its own SQLCA. This requirement also applies to code in a shared library (in a DLL, for example) that uses embedded SQL and is called by more than one thread in your application.

You can use the multiple SQLCA support in any of the supported embedded SQL environments, but it is only required in reentrant code.

You do not need to use multiple SQLCAs to connect to more than one database or have more than one connection to a single database.

Each SQLCA can have one unnamed connection. Each SQLCA has an active or current connection.

All operations on a given database connection must use the same SQLCA that was used when the connection was established.

Note: Operations on different connections are subject to the normal record locking mechanisms and may cause each other to block and possibly to deadlock.

Static and Dynamic SQL

There are two ways to embed SQL statements into a C program:

- Static statements
- Dynamic statements

Static SQL Statements

All standard SQL data manipulation and data definition statements can be embedded in a C program by prefixing them with EXEC SQL and suffixing the statement with a semicolon (;). These statements are referred to as *static* statements.

Static statements can contain references to host variables. Host variables can only be used in place of string or numeric constants. They cannot be used to substitute column names or table names; dynamic statements are required to perform those operations.

Dynamic SQL Statements

In the C language, strings are stored in arrays of characters. Dynamic statements are constructed in C language strings. These statements can then be executed using the PREPARE and EXECUTE statements. These SQL statements cannot reference host variables in the same manner as static statements since the C language variables are not accessible by name when the C program is executing.

To pass information between the statements and the C language variables, a data structure called the *SQL Descriptor Area* (SQLDA) is used. This structure is set up for you by the SQL preprocessor if you specify a list of host variables on the EXECUTE statement in the USING clause. These variables correspond by position to placeholders in the appropriate positions of the prepared statement.

A *placeholder* is put in the statement to indicate where host variables are to be accessed. A placeholder is either a question mark (?) or a host variable reference as in static statements (a host variable name preceded by a colon). In the latter case, the host variable name used in the actual text of the statement serves only as a placeholder indicating a reference to the SQL descriptor area.

A host variable used to pass information to the database is called a *bind variable*.

Example

```
EXEC SQL BEGIN DECLARE SECTION;
char      comm[200];
char      street[30];
char      city[20];
a_sql_len cityind;
long      empnum;
EXEC SQL END DECLARE SECTION;
```

```

...
sprintf( comm,
    "UPDATE %s SET Street = :?, City = :?"
    "WHERE EmployeeID = :?",
    tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
EXEC SQL EXECUTE S1 USING :street, :city:cityind, :empnum;

```

This method requires you to know how many host variables there are in the statement. Usually, this is not the case. So, you can set up your own `SQLDA` structure and specify this `SQLDA` in the `USING` clause on the `EXECUTE` statement.

The `DESCRIBE BIND VARIABLES` statement returns the host variable names of the bind variables that are found in a prepared statement. This makes it easier for a C program to manage the host variables. The general method is as follows:

```

EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
...
sprintf( comm,
    "UPDATE %s SET Street = :street, City = :city"
    " WHERE EmployeeID = :empnum",
    tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
/* Assume that there are no more than 10 host variables.
 * See next example if you cannot put a limit on it. */
sqllda = alloc_sqllda( 10 );
EXEC SQL DESCRIBE BIND VARIABLES FOR S1 INTO sqllda;
/* sqllda->sqlld will tell you how many
 host variables there were. */
/* Fill in SQLDA_VARIABLE fields with
 values based on name fields in sqllda. */
...
EXEC SQL EXECUTE S1 USING DESCRIPTOR sqllda;
free_sqllda( sqllda );

```

SQLDA contents

The `SQLDA` consists of an array of variable descriptors. Each descriptor describes the attributes of the corresponding C program variable or the location that the database stores data into or retrieves data from:

- data type
- length if *type* is a string type
- memory address
- indicator variable

Indicator variables and NULL

The indicator variable is used to pass a `NULL` value to the database or retrieve a `NULL` value from the database. The database server also uses the indicator variable to indicate truncation conditions encountered during a database operation. The indicator variable is set to a positive value when not enough space was provided to receive a database value.

Dynamic SELECT Statement

A SELECT statement that returns only a single row can be prepared dynamically, followed by an EXECUTE with an INTO clause to retrieve the one-row result. SELECT statements that return multiple rows, however, are managed using dynamic cursors.

With dynamic cursors, results are put into a host variable list or a SQLDA that is specified on the FETCH statement (FETCH INTO and FETCH USING DESCRIPTOR). Since the number of SELECT list items is usually unknown, the SQLDA route is the most common. The DESCRIBE SELECT LIST statement sets up a SQLDA with the types of the SELECT list items. Space is then allocated for the values using the fill_sqlda or fill_s_sqlda functions, and the information is retrieved by the FETCH USING DESCRIPTOR statement.

The typical scenario is as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
int actual_size;
SQLDA * sqlda;
...
sprintf( comm, "SELECT * FROM %s", table_name );
EXEC SQL PREPARE S1 FROM :comm;
/* Initial guess of 10 columns in result.
   If it is wrong, it is corrected right
   after the first DESCRIBE by reallocating
   sqlda and doing DESCRIBE again. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE SELECT LIST FOR S1
      INTO sqlda;
if( sqlda->sqlc > sqlda->sqln )
{
    actual_size = sqlda->sqlc;
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( actual_size );
    EXEC SQL DESCRIBE SELECT LIST FOR S1
          INTO sqlda;
}
fill_sqlda( sqlda );
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
EXEC SQL WHENEVER NOTFOUND {break};
for( ;; )
{
    EXEC SQL FETCH C1 USING DESCRIPTOR sqlda;
    /* do something with data */
}
EXEC SQL CLOSE C1;
EXEC SQL DROP STATEMENT S1;
```

Note: To avoid consuming unnecessary resources, ensure that statements are dropped after use.

The SQL Descriptor Area (SQLDA)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file `sqlda.h`.

There are functions in the database interface shared library or DLL that you can use to manage SQLDAs.

When host variables are used with static SQL statements, the preprocessor constructs a SQLDA for those host variables. It is this SQLDA that is actually passed to and from the database server.

The SQLDA Header File

The contents of `sqlda.h` are as follows:

```
#ifndef _SQLDA_H_INCLUDED
#define _SQLDA_H_INCLUDED
#define II_SQLDA
#include "sqlca.h"
#if defined( _SQL_PACK_STRUCTURES )
    #if defined( _MSC_VER ) && _MSC_VER > 800
        #pragma warning(push)
        #pragma warning(disable:4103)
    #endif
    #include "pshpk1.h"
#endif
#define SQL_MAX_NAME_LEN    30
#define _sqldafar
typedef short int a_sql_type;

struct sqlname {
    short int    length; /* length of char data */
    char        data[ SQL_MAX_NAME_LEN ]; /* data */
};

struct sqlvar {
    /* array of variable descriptors */
    short int    sqltype; /* type of host variable */
    a_sql_len    sqllen; /* length of host variable */
    void        *sqldata; /* address of variable */
    a_sql_len    *sqlind; /* indicator variable pointer */
    struct sqlname sqlname;
};

#if defined( _SQL_PACK_STRUCTURES )
    #include "poppk.h"
    /* The SQLDA should be 4-byte aligned */
    #include "pshpk4.h"
#endif
#endif
```

```

struct sqlda {
    unsigned char    sqldaid[8]; /* eye catcher "SQLDA" */
    a_sql_int32      sqldabc; /* length of sqlda structure */
    short int        sqln; /* descriptor size in number of entries */
    short int        sqld; /* number of variables found by DESCRIBE */
    struct sqlvar    sqlvar[1]; /* array of variable descriptors */
};

#define SCALE(sqllen) ((sqllen)/256)
#define PRECISION(sqllen) ((sqllen)&0xff)
#define SET_PRECISION_SCALE(sqllen,precision,scale) \
    sqllen = (scale)*256 + (precision)
#define DECIMALSTORAGE(sqllen) (PRECISION(sqllen)/2 + 1)
typedef struct sqlda SQLDA;
typedef struct sqlvar SQLVAR, SQLDA_VARIABLE;
typedef struct sqlname SQLNAME, SQLDA_NAME;
#ifndef SQLDASIZE
#define SQLDASIZE(n) ( sizeof( struct sqlda ) + \
    (n-1) * sizeof( struct sqlvar ) )
#endif
#ifdef _SQL_PACK_STRUCTURES )
#include "poppk.h"
#ifdef _MSC_VER ) && _MSC_VER > 800
#pragma warning(pop)
#endif
#endif
#endif
#endif

```

SQLDA Fields

The SQLDA fields have the following meanings:

Field	Description
sqldaid	An 8-byte character field that contains the string SQLDA as an identification of the SQLDA structure. This field helps in debugging when you are looking at memory contents.
sqldabc	A 32-bit integer containing the length of the SQLDA structure.
sqln	The number of variable descriptors allocated in the sqlvar array.
sqld	The number of variable descriptors that are valid (contain information describing a host variable). This field is set by the DESCRIBE statement. As well, you can set it when supplying data to the database server.

Field	Description
sqlvar	An array of descriptors of type struct sqlvar, each describing a host variable.

SQLDA Host Variable Descriptions

Each sqlvar structure in the SQLDA describes a host variable. The fields of the sqlvar structure have the following meanings:

- **sqltype** – The type of the variable that is described by this descriptor.

The low order bit indicates whether NULL values are allowed. Valid types and constant definitions can be found in the `sqldef.h` header file.

This field is filled by the DESCRIBE statement. You can set this field to any type when supplying data to the database server or retrieving data from the database server. Any necessary type conversion is done automatically.

- **sqllen** – The length of the variable. A sqllen value has type `a_sql_len`. What the length actually means depends on the type information and how the SQLDA is being used.

For LONG VARCHAR, LONG NVARCHAR, and LONG BINARY data types, the `array_len` field of the DT_LONGVARCHAR, DT_LONGNVARCHAR, or DT_LONGBINARY data type structure is used instead of the sqllen field.

- **sqldata** – A pointer to the memory occupied by this variable. This memory must correspond to the sqltype and sqllen fields.

For UPDATE and INSERT statements, this variable is not involved in the operation if the sqldata pointer is a null pointer. For a FETCH, no data is returned if the sqldata pointer is a null pointer. In other words, the column returned by the sqldata pointer is an *unbound column*.

If the DESCRIBE statement uses LONG NAMES, this field holds the long name of the result set column. If, in addition, the DESCRIBE statement is a DESCRIBE USER TYPES statement, then this field holds the long name of the user-defined data type, instead of the column. If the type is a base type, the field is empty.

- **sqlind** – A pointer to the indicator value. An indicator value has type `a_sql_len`. A negative indicator value indicates a NULL value. A positive indicator value indicates that this variable has been truncated by a FETCH statement, and the indicator value contains the length of the data before truncation. A value of -2 indicates a conversion error if the `conversion_error` database option is set to Off.

If the sqlind pointer is the null pointer, no indicator variable pertains to this host variable.

The sqlind field is also used by the DESCRIBE statement to indicate parameter types. If the type is a user-defined data type, this field is set to DT_HAS_USERTYPE_INFO. In this case, you may want to perform a DESCRIBE USER TYPES to obtain information about the user-defined data types.

- **sqlname** – A VARCHAR-like structure, as follows:

```
struct sqlname {
    short int length;
    char data[ SQL_MAX_NAME_LEN ];
};
```

It is filled by a DESCRIBE statement and is not otherwise used. This field has a different meaning for the two formats of the DESCRIBE statement:

- **SELECT LIST** – The name data buffer is filled with the column heading of the corresponding item in the SELECT list.
- **BIND VARIABLES** – The name data buffer is filled with the name of the host variable that was used as a bind variable, or "?" if an unnamed parameter marker is used.

On a DESCRIBE SELECT LIST statement, any indicator variables present are filled with a flag indicating whether the SELECT list item is updatable or not. More information about this flag can be found in the `sqldef.h` header file.

If the DESCRIBE statement is a DESCRIBE USER TYPES statement, then this field holds the long name of the user-defined data type instead of the column. If the type is a base type, the field is empty.

SQLDA sqlen Field Values

SQLDA sqlen field values after a DESCRIBE, when sending values, and when retrieving data.

SQLDA sqlen Field Values After a DESCRIBE

The DESCRIBE statement gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

The following table indicates the values of the sqlen and sqltype structure members returned by the DESCRIBE statement for the various database types (both SELECT LIST and BIND VARIABLE DESCRIBE statements). For a user-defined database data type, the base type is described.

Your program can use the types and lengths returned from a DESCRIBE, or you may use another type. The database server performs type conversions between any two types. The memory pointed to by the sqldata field must correspond to the sqltype and sqlen fields. The embedded SQL type is obtained by a bitwise AND of sqltype with DT_TYPES (sqltype & DT_TYPES).

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
BIGINT	DT_BIGINT	8
BINARY(n)	DT_BINARY	n

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
BIT	DT_BIT	1
CHAR(n)	DT_FIXCHAR ¹	n times maximum data expansion when converting from database character set to the client's CHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
CHAR(n CHAR)	DT_FIXCHAR ¹	n times maximum character length in the client's CHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
DATE	DT_DATE	length of longest formatted string
DECIMAL(p,s)	DT_DECIMAL	low byte of length field in SQLDA set to p, and high byte set to s. See PRECISION and SCALE macros in sqlda.h.
DOUBLE	DT_DOUBLE	8
FLOAT	DT_FLOAT	4
INT	DT_INT	4
LONG BINARY	DT_LONGBINARY	32767
LONG NVARCHAR	DT_LONGVARCHAR / DT_LONGNVARCHAR ²	32767
LONG VARCHAR	DT_LONGVARCHAR	32767

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
NCHAR(n)	DT_FIXCHAR / DT_NFIX-CHAR ²	n times maximum character length in the client's NCHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONG-NVARCHAR with a length of 32767 bytes.
NVARCHAR(n)	DT_VARCHAR / DT_NVARCHAR ²	n times maximum character length in the client's NCHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONG-NVARCHAR with a length of 32767 bytes.
REAL	DT_FLOAT	4
SMALLINT	DT_SMALLINT	2
TIME	DT_TIME	length of longest formatted string
TIMESTAMP	DT_TIMESTAMP	length of longest formatted string
TINYINT	DT_TINYINT	1
UNSIGNED BIGINT	DT_UNSBIGINT	8
UNSIGNED INT	DT_UNSENT	4
UNSIGNED SMALLINT	DT_UNSSMALLINT	2
VARCHAR(n)	DT_VARCHAR ¹	n times maximum data expansion when converting from database character set to the client's CHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
VARCHAR(n CHAR)	DT_VARCHAR ¹	n times maximum character length in the client's CHAR character set. If this length would be more than 32767, then the embedded SQL type returned is DT_LONGVARCHAR with length 32767.

¹ The type returned for CHAR and VARCHAR may be DT_LONGVARCHAR if the maximum byte length in the client's CHAR character set is greater than 32767 bytes.

² The type returned for NCHAR and NVARCHAR may be DT_LONGNVARCHAR if the maximum byte length in the client's NCHAR character set is greater than 32767 bytes. NCHAR, NVARCHAR, and LONG NVARCHAR are described by default as either DT_FIXCHAR, DT_VARCHAR, or DT_LONGVARCHAR, respectively. If the db_change_nchar_charset function has been called, the types are described as DT_NFIXCHAR, DT_NVARCHAR, and DT_LONGNVARCHAR, respectively.

SQLDA sqlen Field Values when Sending Values

The following table indicates how you specify lengths of values when you supply data to the database server in the SQLDA.

Only the data types displayed in the table are allowed in this case. The DT_DATE, DT_TIME, and DT_TIMESTAMP types are treated the same as DT_STRING when supplying information to the database; the value must be a null-terminated character string in an appropriate date or time of day format.

Embedded SQL data type	Program action to set the length
DT_BIGINT	No action required.
DT_BINARY(n)	Length taken from field in BINARY structure.
DT_BIT	No action required.
DT_DATE	Length determined by terminating null character.
DT_DOUBLE	No action required.
DT_FIXCHAR(n)	Length field in SQLDA determines length of string.
DT_FLOAT	No action required.
DT_INT	No action required.

Embedded SQL data type	Program action to set the length
DT_LONGBINARY	Length field ignored.
DT_LONGNVARCHAR	Length field ignored.
DT_LONGVARCHAR	Length field ignored.
DT_NFIXCHAR(n)	Length field in SQLDA determines length of string.
DT_NSTRING	Length determined by terminating \0. If the ansi_blanks option is On and the database is blank-padded, then the length field in the SQLDA must be set to the length of the buffer containing the value (at least the length of the value plus space for the terminating null character).
DT_NVARCHAR	Length taken from field in NVARCHAR structure.
DT_SMALLINT	No action required.
DT_STRING	Length determined by terminating \0. If the ansi_blanks option is On and the database is blank-padded, then the length field in the SQLDA must be set to the length of the buffer containing the value (at least the length of the value plus space for the terminating null character).
DT_TIME	Length determined by terminating null character.
DT_TIMESTAMP	Length determined by terminating null character.
DT_TIMESTAMP_STRUCT	No action required.
DT_UNSBIGINT	No action required.
DT_UNSENT	No action required.
DT_UNSSMALLINT	No action required.
DT_VARCHAR(n)	Length taken from field in VARCHAR structure.
DT_VARIABLE	Length determined by terminating \0.

SQLDA ssqlen Field Values when Retrieving Data

The following table indicates the values of the length field when you retrieve data from the database using a SQLDA. The ssqlen field is never modified when you retrieve data.

Only the interface data types displayed in the table are allowed in this case. The DT_DATE, DT_TIME, and DT_TIMESTAMP data types are treated the same as DT_STRING when you retrieve information from the database. The value is formatted as a character string in the current date format.

Embedded SQL data type	What the program must set length field to when receiving	How the database returns length information after fetching a value
DT_BIGINT	No action required.	No action required.
DT_BINARY(n)	Maximum length of BINARY structure (n+2). The maximum value for n is 32765.	len field of BINARY structure set to actual length in bytes.
DT_BIT	No action required.	No action required.
DT_DATE	Length of buffer.	null character at end of string.
DT_DOUBLE	No action required.	No action required.
DT_FIXCHAR(n)	Length of buffer, in bytes. The maximum value for n is 32767.	Padded with blanks to length of buffer.
DT_FLOAT	No action required.	No action required.
DT_INT	No action required.	No action required.
DT_LONGBINARY	Length field ignored.	Length field ignored.
DT_LONGNVARCHAR	Length field ignored.	Length field ignored.
DT_LONGVARCHAR	Length field ignored.	Length field ignored.
DT_NFIXCHAR(n)	Length of buffer, in bytes. The maximum value for n is 32767.	Padded with blanks to length of buffer.
DT_NSTRING	Length of buffer.	null character at end of string.
DT_NVARCHAR(n)	Maximum length of NVARCHAR structure (n+2). The maximum value for n is 32765.	len field of NVARCHAR structure set to actual length in bytes of string.
DT_SMALLINT	No action required.	No action required.
DT_STRING	Length of buffer.	null character at end of string.

Embedded SQL data type	What the program must set length field to when receiving	How the database returns length information after fetching a value
DT_TIME	Length of buffer.	null character at end of string.
DT_TIMESTAMP	Length of buffer.	null character at end of string.
DT_TIMESTAMP_STRUCT	No action required.	No action required.
DT_UNSBIGINT	No action required.	No action required.
DT_UNSENT	No action required.	No action required.
DT_UNSSMALLINT	No action required.	No action required.
DT_VARCHAR(n)	Maximum length of VARCHAR structure (n+2). The maximum value for n is 32765.	len field of VARCHAR structure set to actual length in bytes of string.

How to Fetch Data Using Embedded SQL

Fetching data in embedded SQL is done using the SELECT statement. There are two cases:

- **The SELECT statement returns at most one row** – Use an INTO clause to assign the returned values directly to host variables.
- **The SELECT statement may return multiple rows** – Use cursors to manage the rows of the result set.

SELECT Statements That Return at Most One Row

A single row query retrieves at most one row from the database. A single-row query SELECT statement has an INTO clause following the SELECT list and before the FROM clause. The INTO clause contains a list of host variables to receive the value for each SELECT list item. There must be the same number of host variables as there are SELECT list items. The host variables may be accompanied by indicator variables to indicate NULL results.

When the SELECT statement is executed, the database server retrieves the results and places them in the host variables. If the query results contain more than one row, the database server returns an error.

If the query results in no rows being selected, an error is returned indicating that no rows can be found (SQLCODE 100). Errors and warnings are returned in the SQLCA structure.

Example

The following code fragment returns 1 if a row from the Employees table is fetched successfully, 0 if the row doesn't exist, and -1 if an error occurs.

```

EXEC SQL BEGIN DECLARE SECTION;
long      id;
char      name[41];
char      sex;
char      birthdate[15];
a_sql_len ind birthdate;
EXEC SQL END DECLARE SECTION;
...
int find_employee( long employee_id )
{
    id = employee_id;
    EXEC SQL SELECT GivenName ||
        ' ' || Surname, Sex, BirthDate
        INTO :name, :sex,
            :birthdate:ind_birthdate
        FROM Employees
        WHERE EmployeeID = :id;
    if( SQLCODE == SQLE_NOTFOUND )
    {
        return( 0 ); /* employee not found */
    }
    else if( SQLCODE < 0 )
    {
        return( -1 ); /* error */
    }
    else
    {
        return( 1 ); /* found */
    }
}

```

Cursors in Embedded SQL

A cursor is used to retrieve rows from a query that has multiple rows in its result set. A cursor is a handle or an identifier for the SQL query and a position within the result set.

Cursor management in embedded SQL involves of the following steps:

1. Declare a cursor for a particular SELECT statement, using the DECLARE CURSOR statement.
2. Open the cursor using the OPEN statement.
3. Retrieve results one row at a time from the cursor using the FETCH statement.
4. Fetch rows until the Row Not Found warning is returned.
Errors and warnings are returned in the SQLCA structure.
5. Close the cursor, using the CLOSE statement.

By default, cursors are automatically closed at the end of a transaction (on COMMIT or ROLLBACK). Cursors that are opened with a WITH HOLD clause are kept open for subsequent transactions until they are explicitly closed.

The following is a simple example of cursor usage:

```

void print_employees( void )
{

```

Embedded SQL

```
EXEC SQL BEGIN DECLARE SECTION;
char      name[50];
char      sex;
char      birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT GivenName || ' ' || Surname, Sex, BirthDate FROM Employees;
EXEC SQL OPEN C1;
for( ;; )
{
  EXEC SQL FETCH C1 INTO :name, :sex, :birthdate:ind_birthdate;
  if( SQLCODE == SQLE_NOTFOUND )
  {
    break;
  }
  else if( SQLCODE < 0 )
  {
    break;
  }

  if( ind_birthdate < 0 )
  {
    strcpy( birthdate, "UNKNOWN" );
  }
  printf( "Name: %s Sex: %c Birthdate: %s\n", name, sex,
birthdate );
}
EXEC SQL CLOSE C1;
}
```

Cursor positioning

A cursor is positioned in one of three places:

- On a row
- Before the first row
- After the last row

Absolute row from start		Absolute row from end
0	Before first row	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		-3
$n - 1$		-2
n		-1
$n + 1$	After last row	0

When a cursor is opened, it is positioned before the first row. The cursor position can be moved using the FETCH statement. It can be positioned to an absolute position either from the start or from the end of the query results. It can also be moved relative to the current cursor position.

There are special *positioned* versions of the UPDATE and DELETE statements that can be used to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, an error is returned indicating that there is no corresponding row in the cursor.

The PUT statement can be used to insert a row into a cursor.

Cursor positioning problems

Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database server does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear until the cursor is closed and opened again.

With SAP Sybase IQ, this occurs if a temporary table had to be created to open the cursor.

The UPDATE statement can cause a row to move in the cursor. This happens if the cursor has an ORDER BY clause that uses an existing index (a temporary table is not created).

Wide Fetches or Array Fetches

The FETCH statement can be modified to fetch more than one row at a time, which may improve performance. This is called a *wide fetch* or an *array fetch*.

SAP Sybase IQ also supports wide puts and inserts.

To use wide fetches in embedded SQL, include the FETCH statement in your code as follows:

```
EXEC SQL FETCH ... ARRAY nnn
```

where ARRAY *nnn* is the last item of the FETCH statement. The fetch count *nnn* can be a host variable. The number of variables in the SQLDA must be the product of *nnn* and the number of columns per row. The first row is placed in SQLDA variables 0 to (columns per row) - 1, and so on.

Each column must be of the same type in each row of the SQLDA, or a SQLDA_INCONSISTENT error is returned.

The server returns in SQLCOUNT the number of records that were fetched, which is always greater than zero unless there is an error or warning. On a wide fetch, a SQLCOUNT of 1 with no error condition indicates that one valid row has been fetched.

Example

The following example code illustrates the use of wide fetches. You can also find this code in %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\esqlwidefetch\widefetch.sqc.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR { PrintSQLError();
    goto err; };

static void PrintSQLError()
{
    char buffer[200];

    printf( "SQL error %d -- %s\n",
        SQLCODE,
        sqlerror_message( &sqlca,
            buffer,
            sizeof( buffer ) ) );
}

static SQLDA * PrepareSQLDA(
    a_sql_statement_number stat0,
    unsigned width,
    unsigned *cols_per_row )

/* Allocate a SQLDA to be used for fetching from
```

```

the statement identified by "stat0". "width"
rows are retrieved on each FETCH request.
The number of columns per row is assigned to
"cols_per_row". */
{
    int          num_cols;
    unsigned     row, col, offset;
    SQLDA *      sqlda;
    EXEC SQL BEGIN DECLARE SECTION;
    a_sql_statement_number stat;
    EXEC SQL END DECLARE SECTION;
    stat = stat0;
    sqlda = alloc_sqllda( 100 );
    if( sqlda == NULL ) return( NULL );
    EXEC SQL DESCRIBE :stat INTO sqlda;
    *cols_per_row = num_cols = sqlda->sqlc;
    if( num_cols * width > sqlda->sqln )
    {
        free_sqllda( sqlda );
        sqlda = alloc_sqllda( num_cols * width );
        if( sqlda == NULL ) return( NULL );
        EXEC SQL DESCRIBE :stat INTO sqlda;
    }
    // copy first row in SQLDA setup by describe
    // to following (wide) rows
    sqlda->sqlc = num_cols * width;
    offset = num_cols;
    for( row = 1; row < width; row++ )
    {
        for( col = 0;
            col < num_cols;
            col++, offset++ )
        {
            sqlda->sqlvar[offset].sqltype =
                sqlda->sqlvar[col].sqltype;
            sqlda->sqlvar[offset].sqln =
                sqlda->sqlvar[col].sqln;
            // optional: copy described column name
            memcpy( &sqlda->sqlvar[offset].sqlname,
                &sqlda->sqlvar[col].sqlname,
                sizeof( sqlda->sqlvar[0].sqlname ) );
        }
    }
    fill_s_sqllda( sqlda, 40 );
    return( sqlda );
err:
    return( NULL );
}
static void PrintFetchedRows(
    SQLDA * sqlda,
    unsigned cols_per_row )
{
    /* Print rows already wide fetched in the SQLDA */
    long    rows_fetched;
    int     row, col, offset;

```

```

if( SQLCOUNT == 0 )
{
    rows_fetched = 1;
}
else
{
    rows_fetched = SQLCOUNT;
}
printf( "Fetched %d Rows:\n", rows_fetched );
for( row = 0; row < rows_fetched; row++ )
{
    for( col = 0; col < cols_per_row; col++ )
    {
        offset = row * cols_per_row + col;
        printf( " \"%s\"",
            (char *)sqllda->sqlvar[offset].sqldata );
    }
    printf( "\n" );
}
}
static int DoQuery(
    char * query_str0,
    unsigned fetch_width0 )
{
    /* Wide Fetch "query_str0" select statement
     * using a width of "fetch_width0" rows" */
    SQLDA *      sqllda;
    unsigned     cols_per_row;
    EXEC SQL BEGIN DECLARE SECTION;
    a_sql_statement_number stat;
    char *      query_str;
    unsigned     fetch_width;
    EXEC SQL END DECLARE SECTION;

    query_str = query_str0;
    fetch_width = fetch_width0;

    EXEC SQL PREPARE :stat FROM :query_str;
    EXEC SQL DECLARE QCURSOR CURSOR FOR :stat
        FOR READ ONLY;
    EXEC SQL OPEN QCURSOR;
    sqllda = PrepareSQLDA( stat,
        fetch_width,
        &cols_per_row );
    if( sqllda == NULL )
    {
        printf( "Error allocating SQLDA\n" );
        return( SQLE_NO_MEMORY );
    }
    for( ;; )
    {
        EXEC SQL FETCH QCURSOR INTO DESCRIPTOR sqllda
            ARRAY :fetch_width;
        if( SQLCODE != SQLE_NOERROR ) break;
        PrintFetchedRows( sqllda, cols_per_row );
    }
}

```

```

EXEC SQL CLOSE QCURSOR;
EXEC SQL DROP STATEMENT :stat;
free_filled_sqllda( sqllda );
err:
return( SQLCODE );
}
void main( int argc, char *argv[] )
{
/* Optional first argument is a select statement,
 * optional second argument is the fetch width */
char *query_str =
    "SELECT GivenName, Surname FROM Employees";
unsigned fetch_width = 10;

if( argc > 1 )
{
    query_str = argv[1];
    if( argc > 2 )
    {
        fetch_width = atoi( argv[2] );
        if( fetch_width < 2 )
        {
            fetch_width = 2;
        }
    }
}
db_init( &sqlca );
EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";

DoQuery( query_str, fetch_width );

EXEC SQL DISCONNECT;
err:
db_fini( &sqlca );
}

```

Notes on using wide fetches

- In the function PrepareSQLDA, the SQLDA memory is allocated using the alloc_sqllda function. This allows space for indicator variables, rather than using the alloc_sqllda_noind function.
- If the number of rows fetched is fewer than the number requested, but is not zero (at the end of the cursor for example), the SQLDA items corresponding to the rows that were not fetched are returned as NULL by setting the indicator value. If no indicator variables are present, an error is generated (SQLE_NO_INDICATOR: no indicator variable for NULL result).
- If a row being fetched has been updated, generating a SQLE_ROW_UPDATED_WARNING warning, the fetch stops on the row that caused the warning. The values for all rows processed to that point (including the row that caused the warning) are returned. SQLCOUNT contains the number of rows that were fetched,

including the row that caused the warning. All remaining SQLDA items are marked as NULL.

- If a row being fetched has been deleted or is locked, generating a `SQL_NO_CURRENT_ROW` or `SQL_LOCKED` error, `SQLCOUNT` contains the number of rows that were read before the error. This does not include the row that caused the error. The `SQLDA` does not contain values for any of the rows since `SQLDA` values are not returned on errors. The `SQLCOUNT` value can be used to reposition the cursor, if necessary, to read the rows.

How to Send and Retrieve Long Values Using Embedded SQL

The method for sending and retrieving `LONG VARCHAR`, `LONG NVARCHAR`, and `LONG BINARY` values in embedded SQL applications is different from that for other data types. The standard `SQLDA` fields are limited to 32767 bytes of data as the fields holding the length information (`sqlen`, `*sqlind`) are 16-bit values. Changing these values to 32-bit values would break existing applications.

The method of describing `LONG VARCHAR`, `LONG NVARCHAR`, and `LONG BINARY` values is the same as for other data types.

Static SQL structures

Separate fields are used to hold the allocated, stored, and untruncated lengths of `LONG BINARY`, `LONG VARCHAR`, and `LONG NVARCHAR` data types. The static SQL data types are defined in `sqlca.h` as follows:

```
#define DECL_LONGVARCHAR( size )      \
    struct { a_sql_uint32    array_len;  \
             a_sql_uint32    stored_len; \
             a_sql_uint32    untrunc_len; \
             char             array[size+1]; \
    }

#define DECL_LONGNVARCHAR( size )    \
    struct { a_sql_uint32    array_len;  \
             a_sql_uint32    stored_len; \
             a_sql_uint32    untrunc_len; \
             char             array[size+1]; \
    }

#define DECL_LONGBINARY( size )      \
    struct { a_sql_uint32    array_len;  \
             a_sql_uint32    stored_len; \
             a_sql_uint32    untrunc_len; \
             char             array[size]; \
    }
```

Dynamic SQL structures

For dynamic SQL, set the `sqltype` field to `DT_LONGVARCHAR`, `DT_LONGNVARCHAR`, or `DT_LONGBINARY` as appropriate. The associated `LONGVARCHAR`, `LONGNVARCHAR`, and `LONGBINARY` structure is as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32    array_len;
    a_sql_uint32    stored_len;
    a_sql_uint32    untrunc_len;
    char            array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

Structure member definitions

For both static and dynamic SQL structures, the structure members are defined as follows:

- **array_len** – (Sending and retrieving.) The number of bytes allocated for the array part of the structure.
- **stored_len** – (Sending and retrieving.) The number of bytes stored in the array. Always less than or equal to `array_len` and `untrunc_len`.
- **untrunc_len** – (Retrieving only.) The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to `stored_len`. If truncation occurs, this value is larger than `array_len`.

Retrieving LONG Data Using Static SQL

Retrieve a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value using static SQL.

Prerequisites

There are no prerequisites for this task.

Task

1. Declare a host variable of type `DECL_LONGVARCHAR`, `DECL_LONGNVARCHAR`, or `DECL_LONGBINARY`, as appropriate. The `array_len` member is filled in automatically.
2. Retrieve the data using `FETCH`, `GET DATA`, or `EXECUTE INTO`. SAP Sybase IQ sets the following information:
 - **indicator variable** – Negative if the value is NULL, 0 if there is no truncation, otherwise the positive untruncated length in bytes up to a maximum of 32767.
 - **stored_len** – The number of bytes stored in the array. Always less than or equal to `array_len` and `untrunc_len`.

- **untrunc_len** – The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to stored_len. If truncation occurs, this value is larger than array_len.

The LONG data is retrieved using static SQL.

Retrieving LONG Data Using Dynamic SQL

Retrieve a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value using dynamic SQL.

Prerequisites

There are no prerequisites for this task.

Task

1. Set the sqltype field to DT_LONGVARCHAR, DT_LONGNVARCHAR, or DT_LONGBINARY as appropriate.
2. Set the sqldata field to point to the LONGVARCHAR, LONGNVARCHAR, or LONGBINARY host variable structure.

You can use the LONGVARCHARSIZE(*n*), LONGNVARCHARSIZE(*n*), or LONGBINARYSIZE(*n*) macro to determine the total number of bytes to allocate to hold *n* bytes of data in the array field.

3. Set the array_len field of the host variable structure to the number of bytes allocated for the array field.
4. Retrieve the data using FETCH, GET DATA, or EXECUTE INTO. SAP Sybase IQ sets the following information:
 - * **sqlind** – This sqllda field is negative if the value is NULL, 0 if there is no truncation, and is the positive untruncated length in bytes up to a maximum of 32767.
 - **stored_len** – The number of bytes stored in the array. Always less than or equal to array_len and untrunc_len.
 - **untrunc_len** – The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to stored_len. If truncation occurs, this value is larger than array_len.

The LONG data is retrieved using dynamic SQL.

Sending LONG Data Using Static SQL

Send LONG values to the database using static SQL from an embedded SQL application.

Prerequisites

There are no prerequisites for this task.

Task

1. Declare a host variable of type `DECL_LONGVARCHAR`, `DECL_LONGNVARCHAR`, or `DECL_LONGBINARY`, as appropriate.
2. If you are sending `NULL`, set the indicator variable to a negative value.
3. Set the `stored_len` field of the host variable structure to the number of bytes of data in the array field.
4. Send the data by opening the cursor or executing the statement.

The embedded SQL application is ready to send `LONG` values to the database.

Sending LONG Data Using Dynamic SQL

Send `LONG` values to the database using dynamic SQL from an embedded SQL application.

Prerequisites

There are no prerequisites for this task.

Task

1. Set the `sqltype` field to `DT_LONGVARCHAR`, `DT_LONGNVARCHAR`, or `DT_LONGBINARY`, as appropriate.
2. If you are sending `NULL`, set `* sqlind` to a negative value.
3. If you are not sending `NULL`, set the `sqldata` field to point to the `LONGVARCHAR`, `LONGNVARCHAR`, or `LONGBINARY` host variable structure.

You can use the `LONGVARCHARSIZE(n)`, `LONGNVARCHARSIZE(n)`, or `LONGBINARYSIZE(n)` macros to determine the total number of bytes to allocate to hold *n* bytes of data in the array field.

4. Set the `array_len` field of the host variable structure to the number of bytes allocated for the array field.
5. Set the `stored_len` field of the host variable structure to the number of bytes of data in the array field. This must not be more than `array_len`.
6. Send the data by opening the cursor or executing the statement.

The embedded SQL application is ready to send `LONG` values to the database.

Simple Stored Procedures in Embedded SQL

You can create and call stored procedures in embedded SQL.

You can embed a `CREATE PROCEDURE` just like any other data definition statement, such as `CREATE TABLE`. You can also embed a `CALL` statement to execute a stored procedure.

Embedded SQL

The following code fragment illustrates both creating and executing a stored procedure in embedded SQL:

```
EXEC SQL CREATE PROCEDURE pettycash(
  IN Amount DECIMAL(10,2) )
BEGIN
  UPDATE account
  SET balance = balance - Amount
  WHERE name = 'bank';

  UPDATE account
  SET balance = balance + Amount
  WHERE name = 'pettycash expense';
END;
EXEC SQL CALL pettycash( 10.72 );
```

To pass host variable values to a stored procedure or to retrieve the output variables, you prepare and execute a CALL statement. The following code fragment illustrates the use of host variables. Both the USING and INTO clauses are used on the EXECUTE statement.

```
EXEC SQL BEGIN DECLARE SECTION;
double hv_expense;
double hv_balance;
EXEC SQL END DECLARE SECTION;

// Code here
EXEC SQL CREATE PROCEDURE pettycash(
  IN expense DECIMAL(10,2),
  OUT endbalance DECIMAL(10,2) )
BEGIN
  UPDATE account
  SET balance = balance - expense
  WHERE name = 'bank';
  UPDATE account
  SET balance = balance + expense
  WHERE name = 'pettycash expense';

  SET endbalance = ( SELECT balance FROM account
                    WHERE name = 'bank' );
END;

EXEC SQL PREPARE S1 FROM 'CALL pettycash( ?, ? )';
EXEC SQL EXECUTE S1 USING :hv_expense INTO :hv_balance;
```

Stored Procedures with Result Sets

Database procedures can also contain SELECT statements. The procedure is declared using a RESULT clause to specify the number, name, and types of the columns in the result set. Result set columns are different from output parameters. For procedures with result sets, the CALL statement can be used in place of a SELECT statement in the cursor declaration:

```
EXEC SQL BEGIN DECLARE SECTION;
char hv_name[100];
EXEC SQL END DECLARE SECTION;
```

```

EXEC SQL CREATE PROCEDURE female_employees()
  RESULT( name char(50) )
BEGIN
  SELECT GivenName || Surname FROM Employees
  WHERE Sex = 'f';
END;

EXEC SQL PREPARE S1 FROM 'CALL female_employees()';

EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
for(;;)
{
  EXEC SQL FETCH C1 INTO :hv_name;
  if( SQLCODE != SQLE_NOERROR ) break;
  printf( "%s\n", hv_name );
}
EXEC SQL CLOSE C1;

```

In this example, the procedure has been invoked with an `OPEN` statement rather than an `EXECUTE` statement. The `OPEN` statement causes the procedure to execute until it reaches a `SELECT` statement. At this point, `C1` is a cursor for the `SELECT` statement within the database procedure. You can use all forms of the `FETCH` statement (backward and forward scrolling) until you are finished with it. The `CLOSE` statement stops execution of the procedure.

If there had been another statement following the `SELECT` in the procedure, it would not have been executed. To execute statements following a `SELECT`, use the `RESUME` cursor-name statement. The `RESUME` statement either returns the warning `SQLE_PROCEDURE_COMPLETE` or it returns `SQLE_NOERROR` indicating that there is another cursor. The example illustrates a two-select procedure:

```

EXEC SQL CREATE PROCEDURE people()
  RESULT( name char(50) )
BEGIN
  SELECT GivenName || Surname
  FROM Employees;

  SELECT GivenName || Surname
  FROM Customers;
END;

EXEC SQL PREPARE S1 FROM 'CALL people()';
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
while( SQLCODE == SQLE_NOERROR )
{
  for(;;)
  {
    EXEC SQL FETCH C1 INTO :hv_name;
    if( SQLCODE != SQLE_NOERROR ) break;
    printf( "%s\n", hv_name );
  }
  EXEC SQL RESUME C1;
}

```

```
}  
EXEC SQL CLOSE C1;
```

Dynamic cursors for CALL statements

These examples have used static cursors. Full dynamic cursors can also be used for the CALL statement.

The DESCRIBE statement works fully for procedure calls. A DESCRIBE OUTPUT produces a SQLDA that has a description for each of the result set columns.

If the procedure does not have a result set, the SQLDA has a description for each INOUT or OUT parameter for the procedure. A DESCRIBE INPUT statement produces a SQLDA having a description for each IN or INOUT parameter for the procedure.

DESCRIBE ALL

DESCRIBE ALL describes IN, INOUT, OUT, and RESULT set parameters. DESCRIBE ALL uses the indicator variables in the SQLDA to provide additional information.

The DT_PROCEDURE_IN and DT_PROCEDURE_OUT bits are set in the indicator variable when a CALL statement is described. DT_PROCEDURE_IN indicates an IN or INOUT parameter and DT_PROCEDURE_OUT indicates an INOUT or OUT parameter. Procedure RESULT columns have both bits clear.

After a DESCRIBE OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE).

Multiple result sets

If you have a procedure that returns multiple result sets, you must re-describe after each RESUME statement if the result sets change shapes.

You need to describe the cursor, not the statement, to re-describe the current position of the cursor.

Request Management with Embedded SQL

Since a typical embedded SQL application must wait for the completion of each database request before carrying out the next step, an application that uses multiple execution threads can carry on with other tasks.

If you must use a single execution thread, then some degree of multitasking can be accomplished by registering a callback function using the db_register_a_callback function with the DB_CALLBACK_WAIT option. Your callback function is called repeatedly by the interface library while the database server or client library is busy processing your database request.

In your callback function, you cannot start another database request but you can cancel the current request using the `db_cancel_request` function. You can use the `db_is_working` function in your message handlers to determine if you have a database request in progress.

Database Backup with Embedded SQL

The recommended way to backup a database is to use the **BACKUP DATABASE** statement.

The `db_backup` function provides another way to perform an online backup in embedded SQL applications. The SAP Sybase IQ utility also makes use of this function.

You can also interface directly to the SAP Sybase IQ Backup utility using the Database Tools `DBBackup` function.

You should only undertake to write a program using the `db_backup` function if your backup requirements are not satisfied by the any of the other backup methods.

Library Function Reference

The SQL preprocessor generates calls to functions in the interface library or DLL. In addition to the calls generated by the SQL preprocessor, a set of library functions is provided to make database operations easier to perform. Prototypes for these functions are included by the `EXEC SQL INCLUDE SQLCA` statement.

This section contains a reference description of these various functions.

DLL entry points

The DLL entry points are the same except that the prototypes have a modifier appropriate for DLLs.

You can declare the entry points in a portable manner using `_esqlentry_`, which is defined in `sqlca.h`. It resolves to the value `__stdcall`.

alloc_sqlda Function

Allocates a SQLDA with descriptors for *numvar* variables.

Syntax

```
struct sqlda * alloc_sqlda( unsigned numvar );
```

Parameters

- **numvar** – The number of variable descriptors to allocate.

Returns

Pointer to a SQLDA if successful and returns the null pointer if there is not enough memory available.

Remarks

Allocates a SQLDA with descriptors for *numvar* variables. The *sqln* field of the SQLDA is initialized to *numvar*. Space is allocated for the indicator variables, the indicator pointers are set to point to this space, and the indicator value is initialized to zero. A null pointer is returned if memory cannot be allocated. It is recommended that you use this function instead of the `alloc_sqlda_noind` function.

alloc_sqlda_noind Function

Allocates a SQLDA with descriptors for *numvar* variables.

Syntax

```
struct sqlda * alloc_sqlda_noind( unsigned numvar );
```

Parameters

- **numvar** – The number of variable descriptors to allocate.

Returns

Pointer to a SQLDA if successful and returns the null pointer if there is not enough memory available.

Remarks

Allocates a SQLDA with descriptors for *numvar* variables. The *sqln* field of the SQLDA is initialized to *numvar*. Space is not allocated for indicator variables; the indicator pointers are set to the null pointer. A null pointer is returned if memory cannot be allocated.

db_backup Function

Although this function provides one way to add backup features to an application, the recommended way to do this task is to use the **BACKUP DATABASE** statement.

Syntax

```
void db_backup(  
SQLCA * sqlca,  
int op,  
int file_num,  
unsigned long page_num,  
struct sqlda * sqlda);
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **op** – The action or operation to be performed.
- **file_num** – The file number of the database.
- **page_num** – The page number of the database. A value in the range 0 to the maximum number of pages less 1.
- **sqlda** – A pointer to a SQLDA structure.

Authorization

Must be connected as a user with BACKUP DATABASE system privilege, or have the SYS_RUN_REPLICATION_ROLE system role.

Remarks

Although this function provides one way to add backup features to an application, the recommended way to do this task is to use the **BACKUP DATABASE** statement.

The action performed depends on the value of the *op* parameter:

- **DB_BACKUP_START** – Must be called before a backup can start. Only one backup can be running per database at one time against any given database server. Database checkpoints are disabled until the backup is complete (db_backup is called with an *op* value of DB_BACKUP_END). If the backup cannot start, the SQLCODE is SQLE_BACKUP_NOT_STARTED. Otherwise, the SQLCOUNT field of the *sqlca* is set to the database page size. Backups are processed one page at a time.

The *file_num*, *page_num*, and *sqlda* parameters are ignored.

- **DB_BACKUP_OPEN_FILE** – Open the database file specified by *file_num*, which allows pages of the specified file to be backed up using DB_BACKUP_READ_PAGE. Valid file numbers are 0 through DB_BACKUP_MAX_FILE for the root database files, and 0 through DB_BACKUP_TRANS_LOG_FILE for the transaction log file. If the specified file does not exist, the SQLCODE is SQLE_NOTFOUND. Otherwise, SQLCOUNT contains the number of pages in the file, SQLIOESTIMATE contains a 32-bit value (POSIX time_t) that identifies the time that the database file was created, and the operating system file name is in the *sqlerrmc* field of the SQLCA.

The *page_num* and *sqlda* parameters are ignored.

- **DB_BACKUP_READ_PAGE** – Read one page of the database file specified by *file_num*. The *page_num* should be a value from 0 to one less than the number of pages returned in SQLCOUNT by a successful call to db_backup with the DB_BACKUP_OPEN_FILE operation. Otherwise, SQLCODE is set to SQLE_NOTFOUND. The *sqlda* descriptor should be set up with one variable of type DT_BINARY or DT_LONG_BINARY pointing to a buffer. The buffer should be large enough to hold binary data of the size returned in the SQLCOUNT field on the call to db_backup with the DB_BACKUP_START operation.

DT_BINARY data contains a two-byte length followed by the actual binary data, so the buffer must be two bytes longer than the page size.

Note: This call makes a copy of the specified database page into the buffer, but it is up to the application to save the buffer on some backup media.

- **DB_BACKUP_READ_RENAME_LOG** – This action is the same as DB_BACKUP_READ_PAGE, except that after the last page of the transaction log has been returned, the database server renames the transaction log and starts a new one.

If the database server is unable to rename the log at the current time (for example in version 7.0.x or earlier databases there may be incomplete transactions), the SQLE_BACKUP_CANNOT_RENAME_LOG_YET error is set. In this case, do not use the page returned, but instead reissue the request until you receive SQLE_NOERROR and then write the page. Continue reading the pages until you receive the SQLE_NOTFOUND condition.

The SQLE_BACKUP_CANNOT_RENAME_LOG_YET error may be returned multiple times and on multiple pages. In your retry loop, you should add a delay so as not to slow the server down with too many requests.

When you receive the SQLE_NOTFOUND condition, the transaction log has been backed up successfully and the file has been renamed. The name for the old transaction file is returned in the *sqlerrmc* field of the SQLCA.

You should check the *sqlda->sqlvar[0].sqlind* value after a *db_backup* call. If this value is greater than zero, the last log page has been written and the log file has been renamed. The new name is still in *sqlca.sqlerrmc*, but the SQLCODE value is SQLE_NOERROR.

You should not call *db_backup* again after this, except to close files and finish the backup. If you do, you get a second copy of your backed up log file and you receive SQLE_NOTFOUND.

- **DB_BACKUP_CLOSE_FILE** – Must be called when processing of one file is complete to close the database file specified by *file_num*.

The *page_num* and *sqlda* parameters are ignored.

- **DB_BACKUP_END** – Must be called at the end of the backup. No other backup can start until this backup has ended. Checkpoints are enabled again.

The *file_num*, *page_num* and *sqlda* parameters are ignored.

- **DB_BACKUP_PARALLEL_START** – Starts a parallel backup. Like DB_BACKUP_START, only one backup can be running against a database at one time on any given database server. Database checkpoints are disabled until the backup is complete (until *db_backup* is called with an *op* value of DB_BACKUP_END). If the backup cannot start, you receive SQLE_BACKUP_NOT_STARTED. Otherwise, the SQLCOUNT field of the *sqlca* is set to the database page size.

The *file_num* parameter instructs the database server to rename the transaction log and start a new one after the last page of the transaction log has been returned. If the value is

non-zero then the transaction log is renamed or restarted. Otherwise, it is not renamed and restarted. This parameter eliminates the need for the `DB_BACKUP_READ_RENAME_LOG` operation, which is not allowed during a parallel backup operation.

The `page_num` parameter informs the database server of the maximum size of the client's buffer, in database pages. On the server side, the parallel backup readers try to read sequential blocks of pages—this value lets the server know how large to allocate these blocks: passing a value of `nnn` lets the server know that the client is willing to accept at most `nnn` database pages at a time from the server. The server may return blocks of pages of less than size `nnn` if it is unable to allocate enough memory for blocks of `nnn` pages. If the client does not know the size of database pages until after the call to `DB_BACKUP_PARALLEL_START`, this value can be provided to the server with the `DB_BACKUP_INFO` operation. This value must be provided before the first call to retrieve backup pages (`DB_BACKUP_PARALLEL_READ`).

Note: If you are using `db_backup` to start a parallel backup, `db_backup` does not create writer threads. The caller of `db_backup` must receive the data and act as the writer.

- **DB_BACKUP_INFO** – This parameter provides additional information to the database server about the parallel backup. The `file_num` parameter indicates the type of information being provided, and the `page_num` parameter provides the value. You can specify the following additional information with `DB_BACKUP_INFO`:
 - **DB_BACKUP_INFO_PAGES_IN_BLOCK** – The `page_num` argument contains the maximum number of pages that should be sent back in one block.
 - **DB_BACKUP_INFO_CHKPT_LOG** – This is the client-side equivalent to the `WITH CHECKPOINT LOG` option of the **BACKUP DATABASE** statement. A `page_num` value of `DB_BACKUP_CHKPT_COPY` indicates `COPY`, while the value `DB_BACKUP_CHKPT_NOCOPY` indicates `NO COPY`. If this value is not provided it defaults to `COPY`.
- **DB_BACKUP_PARALLEL_READ** – This operation reads a block of pages from the database server. Before invoking this operation, use the `DB_BACKUP_OPEN_FILE` operation to open all the files that you want to back up. `DB_BACKUP_PARALLEL_READ` ignores the `file_num` and `page_num` arguments.

The `sqllda` descriptor should be set up with one variable of type `DT_LONGBINARY` pointing to a buffer. The buffer should be large enough to hold binary data of the size `nnn` pages (specified in the `DB_BACKUP_START_PARALLEL` operation, or in a `DB_BACKUP_INFO` operation).

The server returns a sequential block of database pages for a particular database file. The page number of the first page in the block is returned in the `SQLCOUNT` field. The file number that the pages belong to is returned in the `SQLIOESTIMATE` field, and this value matches one of the file numbers used in the `DB_BACKUP_OPEN_FILE` calls. The size of the data returned is available in the `stored_len` field of the `DT_LONGBINARY` variable, and is always a multiple of the database page size. While the data returned by this call contains a block of sequential pages for a given file, it is not safe to assume that separate

blocks of data are returned in sequential order, or that all of one database file's pages are returned before another database file's pages. The caller should be prepared to receive portions of another individual file out of sequential order, or of any opened database file on any given call.

An application should make repeated calls to this operation until the size of the read data is 0, or the value of `sqlda->sqlvar[0].sqlind` is greater than 0. If the backup is started with transaction log renaming/restarting, `SQLERROR` could be set to `SQLE_BACKUP_CANNOT_RENAME_LOG_YET`. In this case, do not use the pages returned, but instead reissue the request until you receive `SQLE_NOERROR`, and then write the data. The `SQLE_BACKUP_CANNOT_RENAME_LOG_YET` error may be returned multiple times and on multiple pages. In your retry loop, you should add a delay so the database server is not slowed down by too many requests. Continue reading the pages until either of the first two conditions are met.

The `dbbackup` utility uses the following algorithm. This is not C code, and does not include error checking.

```
sqlda->sqlind = 1;
sqlda->sqlvar[0].sqltype = DT_LONGBINARY

/* Allocate LONGBINARY value for page buffer. It MUST have */
/* enough room to hold the requested number (128) of database pages */
sqlda->sqlvar[0].sqldata = allocated buffer

/* Open the server files needing backup */
for file_num = 0 to DB_BACKUP_MAX_FILE
  db_backup( ... DB_BACKUP_OPEN_FILE, file_num ... )
  if SQLCODE == SQLE_NO_ERROR
    /* The file exists */
    num_pages = SQLCOUNT
    file_time = SQLE_IO_ESTIMATE
    open backup file with name from sqlca.sqlerrmc
  end for

/* read pages from the server, write them locally */
while TRUE
  /* file_no and page_no are ignored */
  db_backup( &sqlca, DB_BACKUP_PARALLEL_READ, 0, 0, &sqlda );

  if SQLCODE != SQLE_NO_ERROR
    break;

  if buffer->stored_len == 0 || sqlda->sqlvar[0].sqlind > 0
    break;

  /* SQLCOUNT contains the starting page number of the block */
  /* SQLIOESTIMATE contains the file number the pages belong to */
  write block of pages to appropriate backup file
end while

/* close the server backup files */
for file_num = 0 to DB_BACKUP_MAX_FILE
```

```

    /* close backup file */
    db_backup( ... DB_BACKUP_CLOSE_FILE, file_num ... )
end for

/* shut down the backup */
db_backup( ... DB_BACKUP_END ... )

/* cleanup */
free page buffer

```

db_cancel_request Function

Cancels the currently active database server request. This function checks to make sure a database server request is active before sending the cancel request.

Syntax

```
int db_cancel_request( SQLCA * sqlca );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Returns

1 when the cancel request is sent; 0 if no request is sent.

Remarks

A non-zero return value does not mean that the request was canceled. There are a few critical timing cases where the cancel request and the response from the database or server cross. In these cases, the cancel simply has no effect, even though the function still returns TRUE.

The `db_cancel_request` function can be called asynchronously. This function and `db_is_working` are the only functions in the database interface library that can be called asynchronously using a SQLCA that might be in use by another request.

If you cancel a request that is carrying out a cursor operation, the position of the cursor is indeterminate. You must locate the cursor by its absolute position or close it, following the cancel.

db_change_char_charset Function

Changes the application's CHAR character set for this connection.

Syntax

```
unsigned int db_change_char_charset(
SQLCA * sqlca,
char * charset );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **charset** – A string representing the character set.

Returns

1 if the change is successful; 0 otherwise.

Remarks

Data sent and fetched using DT_FIXCHAR, DT_VARCHAR, DT_LONGVARCHAR, and DT_STRING types are in the CHAR character set.

db_change_nchar_charset Function

Changes the application's NCHAR character set for this connection.

Syntax

```
unsigned int db_change_nchar_charset (  
SQLCA * sqlca,  
char * charset );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **charset** – A string representing the character set.

Returns

1 if the change is successful; 0 otherwise.

Remarks

Data sent and fetched using DT_NFIXCHAR, DT_NVARCHAR, DT_LONGNVARCHAR, and DT_NSTRING host variable types are in the NCHAR character set.

If the db_change_nchar_charset function is not called, all data is sent and fetched using the CHAR character set. Typically, an application that wants to send and fetch Unicode data should set the NCHAR character set to UTF-8.

If this function is called, the charset parameter is usually "UTF-8". The NCHAR character set cannot be set to UTF-16.

In embedded SQL, NCHAR, NVARCHAR and LONG NVARCHAR are described as DT_FIXCHAR, DT_VARCHAR, and DT_LONGVARCHAR, respectively, by default. If the db_change_nchar_charset function has been called, these types are described as DT_NFIXCHAR, DT_NVARCHAR, and DT_LONGNVARCHAR, respectively.

db_find_engine Function

Returns status information about the local database server.

Syntax

```
unsigned short db_find_engine(
SQLCA * sqlca,
char * name );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **name** – NULL or a string containing the server's name.

Returns

Server status as an unsigned short value, or 0 if no server can be found over shared memory.

Remarks

Returns an unsigned short value, which indicates status information about the local database server whose name is *name*. If no server can be found over shared memory with the specified name, the return value is 0. A non-zero value indicates that the local server is currently running.

If a null pointer is specified for *name*, information is returned about the default database server.

Each bit in the return value conveys some information. Constants that represent the bits for the various pieces of information are defined in the `sqldef.h` header file. Their meaning is described below.

- **DB_ENGINE** – This flag is always set.
- **DB_CLIENT** – This flag is always set.
- **DB_CAN_MULTI_DB_NAME** – This flag is obsolete.
- **DB_DATABASE_SPECIFIED** – This flag is always set.
- **DB_ACTIVE_CONNECTION** – This flag is always set.
- **DB_CONNECTION_DIRTY** – This flag is obsolete.
- **DB_CAN_MULTI_CONNECT** – This flag is obsolete.
- **DB_NO_DATABASES** – This flag is set if the server has no databases started.

db_fini Function

This function frees resources used by the database interface or DLL.

Syntax

```
int db_fini( SQLCA * sqlca );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Returns

Non-zero value for success; 0 otherwise.

Remarks

You must not make any other library calls or execute any embedded SQL statements after `db_fini` is called. If an error occurs during processing, the error code is in SQLCA and the function returns 0. If there are no errors, a non-zero value is returned.

You need to call `db_fini` once for each SQLCA being used.

The `db_fini` function should not be called directly or indirectly from the `DIIMain` function in a Windows Dynamic Link Library. The `DIIMain` entry point function is intended to perform only simple initialization and termination tasks. Calling `db_fini` can create deadlocks and circular dependencies.

db_get_property Function

Obtains information about the database interface or the server to which you are connected.

Syntax

```
unsigned int db_get_property(  
SQLCA * sqlca,  
a_db_property property,  
char * value_buffer,  
int value_buffer_size );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **a_db_property** – The property requested, either `DB_PROP_CLIENT_CHARSET`, `DB_PROP_SERVER_ADDRESS`, or `DB_PROP_DBLIB_VERSION`.
- **value_buffer** – This argument is filled with the property value as a null-terminated string.
- **value_buffer_size** – The maximum length of the string `value_buffer`, including room for the terminating null character.

Returns

1 if successful; 0 otherwise.

Remarks

The following properties are supported:

- **DB_PROP_CLIENT_CHARSET** – This property value gets the client character set (for example, "windows-1252").
- **DB_PROP_SERVER_ADDRESS** – This property value gets the current connection's server network address as a printable string. The shared memory protocol always returns the empty string for the address. The TCP/IP protocol returns non-empty string addresses.
- **DB_PROP_DBLIB_VERSION** – This property value gets the database interface library's version (for example, "16.0.0.1297").

db_init Function

This function initializes the database interface library.

Syntax

```
int db_init( SQLCA * sqlca );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Returns

Non-zero value if successful; 0 otherwise.

Remarks

This function must be called before any other library call is made and before any embedded SQL statement is executed. The resources the interface library required for your program are allocated and initialized on this call.

Use `db_fini` to free the resources at the end of your program. If there are any errors during processing, they are returned in the SQLCA and 0 is returned. If there are no errors, a non-zero value is returned and you can begin using embedded SQL statements and functions.

Usually, this function should be called only once (passing the address of the global `sqlca` variable defined in the `sqlca.h` header file). If you are writing a DLL or an application that has multiple threads using embedded SQL, call `db_init` once for each SQLCA that is being used.

db_is_working Function

Returns 1 if your application has a database request in progress that uses the given `sqlca` and 0 if there is no request in progress that uses the given `sqlca`.

Syntax

```
unsigned short db_is_working( SQLCA * sqlca );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Returns

1 if your application has a database request in progress that uses the given *sqlca* and 0 if there is no request in progress that uses the given *sqlca*.

Remarks

This function can be called asynchronously. This function and `db_cancel_request` are the only functions in the database interface library that can be called asynchronously using a SQLCA that might be in use by another request.

db_locate_servers Function

Provides programmatic access to the information displayed by the `dblocate` utility, listing all the SAP Sybase IQ database servers on the local network that are listening on TCP/IP.

Syntax

```
unsigned int db_locate_servers(  
SQLCA * sqlca,  
SQL_CALLBACK_PARM callback_address,  
void * callback_user_data );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **callback_address** – The address of a callback function.
- **callback_user_data** – The address of a user-defined area in which to store data.

Returns

1 if successful; 0 otherwise.

Remarks

The callback function must have the following prototype:

```
int (*)( SQLCA * sqlca,  
a_server_address * server_addr,  
void * callback_user_data );
```

The callback function is called for each server found. If the callback function returns 0, `db_locate_servers` stops iterating through servers.

The `sqlca` and `callback_user_data` passed to the callback function are those passed into `db_locate_servers`. The second parameter is a pointer to an `a_server_address` structure. `a_server_address` is defined in `sqlca.h`, with the following definition:

```
typedef struct a_server_address {  
a_sql_uint32 port_type;
```



```

    a_sql_uint32 port_num;
    char          *name;
    char          *address;
} a_server_address;

```

- **port_type** – Is always PORT_TYPE_TCP at this time (defined to be 6 in `sqlca.h`).
- **port_num** – Is the TCP port number on which this server is listening.
- **name** – Points to a buffer containing the server name.
- **address** – Points to a buffer containing the IP address of the server.

db_locate_servers_ex Function

Provides programmatic access to the information displayed by the `dblocate` utility, listing all the SAP Sybase IQ database servers on the local network that are listening on TCP/IP, and provides a mask parameter used to select addresses passed to the callback function.

Syntax

```

unsigned int db_locate_servers_ex(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data,
unsigned int bitmask);

```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **callback_address** – The address of a callback function.
- **callback_user_data** – The address of a user-defined area in which to store data.
- **bitmask** – A mask composed of any of DB_LOOKUP_FLAG_NUMERIC, DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT, or DB_LOOKUP_FLAG_DATABASES.

Returns

1 if successful; 0 otherwise.

Remarks

The callback function must have the following prototype:

```

int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );

```

The callback function is called for each server found. If the callback function returns 0, `db_locate_servers_ex` stops iterating through servers.

The `sqlca` and `callback_user_data` passed to the callback function are those passed into `db_locate_servers`. The second parameter is a pointer to an `a_server_address` structure. `a_server_address` is defined in `sqlca.h`, with the following definition:

```
typedef struct a_server_address {
    a_sql_uint32    port_type;
    a_sql_uint32    port_num;
    char            *name;
    char            *address;
    char            *dbname;
} a_server_address;
```

- **port_type** – Is always PORT_TYPE_TCP at this time (defined to be 6 in `sqlca.h`).
- **port_num** – Is the TCP port number on which this server is listening.
- **name** – Points to a buffer containing the server name.
- **address** – Points to a buffer containing the IP address of the server.
- **dbname** – Points to a buffer containing the database name.

Three bitmask flags are supported:

```
DB_LOOKUP_FLAG_NUMERIC
DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
DB_LOOKUP_FLAG_DATABASES
```

These flags are defined in `sqlca.h` and can be ORed together.

DB_LOOKUP_FLAG_NUMERIC ensures that addresses passed to the callback function are IP addresses, instead of host names.

DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT specifies that the address includes the TCP/IP port number in the `a_server_address` structure passed to the callback function.

DB_LOOKUP_FLAG_DATABASES specifies that the callback function is called once for each database found, or once for each database server found if the database server doesn't support sending database information (version 9.0.2 and earlier database servers).

db_register_a_callback Function

This function registers callback functions.

Syntax

```
void db_register_a_callback(
    SQLCA * sqlca,
    a_db_callback_index index,
    ( SQL_CALLBACK_PARM ) callback );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **index** – An index value identifying the type of callback as described below.
- **callback** – The address of a user-defined callback function.

Remarks

If you do not register a `DB_CALLBACK_WAIT` callback, the default action is to do nothing. Your application blocks, waiting for the database response. You must register a callback for the `MESSAGE TO CLIENT` statement.

To remove a callback, pass a null pointer as the *callback* function.

The following values are allowed for the *index* parameter:

- **DB_CALLBACK_DEBUG_MESSAGE** – The supplied function is called once for each debug message and is passed a null-terminated string containing the text of the debug message. A debug message is a message that is logged to the LogFile file. In order for a debug message to be passed to this callback, the LogFile connection parameter must be used. The string normally has a newline character (`\n`) immediately before the terminating null character. The prototype of the callback function is as follows:

```
void SQL_CALLBACK debug_message_callback(
SQLCA * sqlca,
char * message_string );
```

- **DB_CALLBACK_START** – The prototype is as follows:

```
void SQL_CALLBACK start_callback( SQLCA * sqlca );
```

This function is called just before a database request is sent to the server.

`DB_CALLBACK_START` is used only on Windows.

- **DB_CALLBACK_FINISH** – The prototype is as follows:

```
void SQL_CALLBACK finish_callback( SQLCA * sqlca );
```

This function is called after the response to a database request has been received by the DBLIB interface DLL. `DB_CALLBACK_FINISH` is used only on Windows operating systems.

- **DB_CALLBACK_CONN_DROPPED** – The prototype is as follows:

```
void SQL_CALLBACK conn_dropped_callback (
SQLCA * sqlca,
char * conn_name );
```

This function is called when the database server is about to drop a connection because of a liveness timeout, through a `DROP CONNECTION` statement, or because the database server is being shut down. The connection name *conn_name* is passed in to allow you to distinguish between connections. If the connection was not named, it has a value of `NULL`.

- **DB_CALLBACK_WAIT** – The prototype is as follows:

```
void SQL_CALLBACK wait_callback( SQLCA * sqlca );
```

This function is called repeatedly by the interface library while the database server or client library is busy processing your database request.

You would register this callback as follows:

```
db_register_a_callback( &sqlca,
    DB_CALLBACK_WAIT,
    (SQL_CALLBACK_PARM)&db_wait_request );
```

- **DB_CALLBACK_MESSAGE** – This is used to enable the application to handle messages received from the server during the processing of a request. Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

The callback prototype is as follows:

```
void SQL_CALLBACK message_callback(
    SQLCA * sqlca,
    unsigned char msg_type,
    an_sql_code code,
    unsigned short length,
    char * msg
);
```

The *msg_type* parameter states how important the message is. You may want to handle different message types in different ways. The following possible values for *msg_type* are defined in `sqldef.h`.

- **MESSAGE_TYPE_INFO** – The message type was INFO.
- **MESSAGE_TYPE_WARNING** – The message type was WARNING.
- **MESSAGE_TYPE_ACTION** – The message type was ACTION.
- **MESSAGE_TYPE_STATUS** – The message type was STATUS.
- **MESSAGE_TYPE_PROGRESS** – The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

The *code* field may provide a SQLCODE associated with the message, otherwise the value is 0. The *length* field tells you how long the message is. The message is not null-terminated. SAP Sybase IQ DBLIB and ODBC clients can use the DB_CALLBACK_MESSAGE parameter to receive progress messages.

For example, the Interactive SQL callback displays STATUS and INFO message on the Messages tab, while messages of type ACTION and WARNING go to a window. If an application does not register this callback, there is a default callback, which causes all messages to be written to the server logfile (if debugging is on and a logfile is specified). In addition, messages of type MESSAGE_TYPE_WARNING and MESSAGE_TYPE_ACTION are more prominently displayed, in an operating system-dependent manner.

When a message callback is not registered by the application, messages sent to the client are saved to the log file when the LogFile connection parameter is specified. Also, ACTION or STATUS messages sent to the client appear in a window on Windows operating systems and are logged to stderr on Unix operating systems.

- **DB_CALLBACK_VALIDATE_FILE_TRANSFER** – This is used to register a file transfer validation callback function. Before allowing any transfer to take place, the client

library will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the client library will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below.

The callback prototype is as follows:

```
int SQL_CALLBACK file_transfer_callback(
SQLCA * sqlca,
char * file_name,
int is_write
);
```

The *file_name* parameter is the name of the file to be read or written. The *is_write* parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the embedded SQL client library allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. If the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

db_start_database Function

Starts the database on an existing server, if possible. Otherwise, a new server is started.

Syntax

```
unsigned int db_start_database( SQLCA * sqlca, char * parms );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **parms** – A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=***value*. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

The database is started on an existing server, if possible. Otherwise, a new server is started.

If the database was already running or was successfully started, the return value is true (non-zero) and SQLCODE is set to 0. Error information is returned in the SQLCA.

If a user ID and password are supplied in the parameters, they are ignored.

The privilege required to start and stop a database is set on the server command line using the -gd option.

db_start_engine Function

Starts the database server if it is not running.

Syntax

```
unsigned int db_start_engine( SQLCA * sqlca, char * parms );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **parms** – A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=value**. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

If the database server was already running or was successfully started, the return value is TRUE (non-zero) and SQLCODE is set to 0. Error information is returned in the SQLCA.

The following call to `db_start_engine` starts the database server, loads the specified database, and names the server `demo`.

```
db_start_engine( &sqlca, "DBF=demo.db;START=iqsrv16" );
```

Unless the ForceStart (FORCE) connection parameter is used and set to YES, the `db_start_engine` function attempts to connect to a server before starting one, to avoid attempting to start a server that is already running.

When the ForceStart connection is set to YES, there is no attempt to connect to a server before trying to start one. This enables the following pair of commands to work as expected:

1. Start a database server named `server_1`:

```
iqsrv16 -n server_1 demo.db
```

2. Force a new server to start and connect to it:

```
db_start_engine( &sqlca,  
  "START=iqsrv16 -n server_2 mydb.db;ForceStart=YES" )
```

If ForceStart (FORCE) is not used and the ServerName (Server) parameter is not used, then the second command would have attempted to connect to server_1. The db_start_engine function does not pick up the server name from the -n option of the StartLine (START) parameter.

db_stop_database Function

Stop the database identified by DatabaseName (DBN) on the server identified by ServerName (Server). If ServerName is not specified, the default server is used.

Syntax

```
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **parms** – A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=***value*. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

By default, this function does not stop a database that has existing connections. If Unconditional (UNC) is set to *yes*, the database is stopped regardless of existing connections.

A return value of TRUE indicates that there were no errors.

The privilege required to start and stop a database is set on the server command line using the -gd option.

db_stop_engine Function

Stops execution of the database server.

Syntax

```
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Embedded SQL

- **parms** – A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=***value*. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

The steps carried out by this function are:

- Look for a local database server that has a name that matches the **ServerName** (Server) parameter. If no **ServerName** is specified, look for the default local database server.
- If no matching server is found, this function returns with success.
- Send a request to the server to tell it to checkpoint and shut down all databases.
- Unload the database server.

By default, this function does not stop a database server that has existing connections. If the **Unconditional=yes** connection parameter is specified, the database server is stopped regardless of existing connections.

A C program can use this function instead of spawning `dbstop`. A return value of **TRUE** indicates that there were no errors.

The use of `db_stop_engine` is subject to the privileges set with the `-gk` server option.

db_string_connect Function

Provides extra functionality beyond the embedded SQL **CONNECT** statement.

Syntax

```
unsigned int db_string_connect( SQLCA * sqlca, char * parms );
```

Parameters

- **sqlca** – A pointer to a **SQLCA** structure.
- **parms** – A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=***value*. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

The return value is **TRUE** (non-zero) if a connection was successfully established and **FALSE** (zero) otherwise. Error information for starting the server, starting the database, or connecting is returned in the **SQLCA**.

db_string_disconnect Function

This function disconnects the connection identified by the ConnectionName parameter. All other parameters are ignored.

Syntax

```
unsigned int db_string_disconnect(
    SQLCA * sqlca,
    char * parms );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **parms** – A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=*value*. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

If no ConnectionName parameter is specified in the string, the unnamed connection is disconnected. This is equivalent to the embedded SQL DISCONNECT statement. The return value is TRUE if a connection was successfully ended. Error information is returned in the SQLCA.

This function shuts down the database if it was started with the AutoStop=yes connection parameter and there are no other connections to the database. It also stops the server if it was started with the AutoStop=yes parameter and there are no other databases running.

db_string_ping_server Function

This function can be used to determine if a server can be located, and optionally, if it a successful connection to a database can be made.

Syntax

```
unsigned int db_string_ping_server(
    SQLCA * sqlca,
    char * connect_string,
    unsigned int connect_to_db );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Embedded SQL

- **connect_string** – The *connect_string* is a normal connection string that may or may not contain server and database information.
- **connect_to_db** – If *connect_to_db* is non-zero (TRUE), then the function attempts to connect to a database on a server. It returns TRUE only if the connection string is sufficient to connect to the named database on the named server.

If *connect_to_db* is zero, then the function only attempts to locate a server. It returns TRUE only if the connection string is sufficient to locate a server. It makes no attempt to connect to the database.

Returns

TRUE (non-zero) if the server or database was successfully located; FALSE (zero) otherwise. Error information for locating the server or database is returned in the SQLCA.

db_time_change Function

This function permits clients to notify the server that the time has changed on the client.

Syntax

```
unsigned int db_time_change(  
SQLCA * sqlca);
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.

Returns

TRUE if successful; FALSE otherwise.

Remarks

This function recalculates the time zone adjustment and sends it to the server. On Windows platforms, it is recommended that applications call this function when they receive the WM_TIMECHANGE message. This will make sure that UTC timestamps are consistent over time changes, time zone changes, or daylight savings time changeovers.

fill_s_sqlda Function

The same as fill_sqlda, except that it changes all the data types in *sqlda* to type DT_STRING.

Syntax

```
struct sqlda * fill_s_sqlda(  
struct sqlda * sqlda,  
unsigned int maxlen);
```

Parameters

- **sqlda** – A pointer to a SQLDA structure.
- **maxlen** – The maximum number of bytes to allocate for the string.

Returns

sqlda if successful and returns NULL if there is not enough memory available.

Remarks

Enough space is allocated to hold the string representation of the type originally specified by the SQLDA, up to a maximum of *maxlen* bytes. The length fields in the SQLDA (*sqlen*) are modified appropriately.

The SQLDA should be freed using the `free_filled_sqlda` function.

fill_sqlda Function

Allocates space for each variable described in each descriptor of *sqlda*, and assigns the address of this memory to the *sqldata* field of the corresponding descriptor.

Syntax

```
struct sqlda * fill_sqlda( struct sqlda * sqlda );
```

Parameters

- **sqlda** – A pointer to a SQLDA structure.

Returns

sqlda if successful and returns NULL if there is not enough memory available.

Remarks

Enough space is allocated for the database type and length indicated in the descriptor.

The SQLDA should be freed using the `free_filled_sqlda` function.

fill_sqlda_ex Function

Allocates space for each variable described in each descriptor of *sqlda*, and assigns the address of this memory to the *sqldata* field of the corresponding descriptor.

Syntax

```
struct sqlda * fill_sqlda_ex( struct sqlda * sqlda , unsigned int flags );
```

Parameters

- **sqlda** – A pointer to a SQLDA structure.
- **flags** – 0 or FILL_SQLDA_FLAG_RETURN_DT_LONG

Returns

sqlda if successful and returns NULL if there is not enough memory available.

Remarks

Enough space is allocated for the database type and length indicated in the descriptor.

The SQLDA should be freed using the `free_filled_sqlda` function.

One flag bit is supported: FILL_SQLDA_FLAG_RETURN_DT_LONG. This flag is defined in `sqlca.h`.

FILL_SQLDA_FLAG_RETURN_DT_LONG preserves DT_LONGVARCHAR, DT_LONGNVARCHAR and DT_LONGBINARY types in the filled descriptor. If this flag bit is not specified, `fill_sqlda_ex` converts DT_LONGVARCHAR, DT_LONGNVARCHAR and DT_LONGBINARY types to DT_VARCHAR, DT_NVARCHAR and DT_BINARY respectively. Using DT_LONGxyz types makes it possible to fetch 32767 bytes, not the 32765 bytes that DT_VARCHAR, DT_NVARCHAR and DT_BINARY are limited to.

`fill_sqlda(sqlda)` is equivalent to `fill_sqlda_ex(sqlda, 0)`.

free_filled_sqlda Function

Free the memory allocated to each `sqldata` pointer and the space allocated for the SQLDA itself. Any null pointer is not freed.

Syntax

```
void free_filled_sqlda( struct sqlda * sqlda );
```

Parameters

- **sqlda** – A pointer to a SQLDA structure.

Remarks

This should only be called if `fill_sqlda`, `fill_sqlda_ex`, or `fill_s_sqlda` was used to allocate the `sqldata` fields of the SQLDA.

Calling this function causes `free_sqlda` to be called automatically, and so any descriptors allocated by `alloc_sqlda` are freed.

free_sqllda Function

Free space allocated to this *sqllda* and free the indicator variable space, as allocated in *fill_sqllda*.

Syntax

```
void free_sqllda( struct sqllda * sqllda );
```

Parameters

- **sqllda** – A pointer to a SQLDA structure.

Remarks

Do not free the memory referenced by each *sqldata* pointer.

free_sqllda_noind Function

Free space allocated to this *sqllda*. Do not free the memory referenced by each *sqldata* pointer. The indicator variable pointers are ignored.

Syntax

```
void free_sqllda_noind( struct sqllda * sqllda );
```

Parameters

- **sqllda** – A pointer to a SQLDA structure.

sql_needs_quotes Function

This function formulates a request to the database server to determine if quotes are needed. Relevant information is stored in the *sqlcode* field.

Syntax

```
unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **str** – A string of characters that is a candidate for a SQL identifier.

Returns

TRUE or FALSE indicating whether the string requires double quotes around it when it is used as a SQL identifier.

Remarks

There are three cases of return value/code combinations:

- **return = FALSE, sqlcode = 0** – The string does not need quotes.
- **return = TRUE** – The sqlcode is always SQLE_WARNING, and the string requires quotes.
- **return = FALSE** – If sqlcode is something other than 0 or SQLE_WARNING, the test is inconclusive.

sqlda_storage Function

An unsigned 32-bit integer value representing the amount of storage required to store any value for the varno variable.

Syntax

```
a_sql_uint32 sqlda_storage( struct sqlda * sqlda, int varno );
```

Parameters

- **sqlda** – A pointer to a SQLDA structure.
- **varno** – An index for a sqlvar host variable.

Returns

An unsigned 32-bit integer value representing the amount of storage required to store any value for the variable.

sqlda_string_length Function

Returns an unsigned 32-bit integer value representing the length of the C string (type DT_STRING) that would be required to hold the variable sqlda->sqlvar[varno] (no matter what its type is).

Syntax

```
a_sql_uint32 sqlda_string_length( struct sqlda * sqlda, int varno );
```

Parameters

- **sqlda** – A pointer to a SQLDA structure.
- **varno** – An index for a sqlvar host variable.

Returns

An unsigned 32-bit integer value representing the length of the C string (type DT_STRING) that would be required to hold the variable sqlda->sqlvar[varno] (no matter what its type is).

sqlerror_message Function

Returns a pointer to a string that contains an error message. The error message contains text for the error code in the SQLCA. If no error was indicated, a null pointer is returned. The error message is placed in the buffer supplied, truncated to length *max* if necessary.

Syntax

```
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

Parameters

- **sqlca** – A pointer to a SQLCA structure.
- **buffer** – The buffer in which to place the message (up to *max* characters).
- **max** – The maximum length of the buffer.

Returns

A pointer to a string that contains an error message or NULL if no error was indicated.

Embedded SQL Statement Summary

ALL embedded SQL statements must be preceded with EXEC SQL and end with a semicolon (;).

There are two groups of embedded SQL statements. Standard SQL statements are used by simply placing them in a C program enclosed with EXEC SQL and a semicolon (;). CONNECT, DELETE, SELECT, SET, and UPDATE have additional formats only available in embedded SQL. The additional formats fall into the second category of embedded SQL specific statements.

Several SQL statements are specific to embedded SQL and can only be used in a C program.

Standard data manipulation and data definition statements can be used from embedded SQL applications. In addition, the following statements are specifically for embedded SQL programming:

- **ALLOCATE DESCRIPTOR statement [ESQL]** – allocate memory for a descriptor.
- **CLOSE statement [ESQL] [SP]** – close a cursor.
- **CONNECT statement [ESQL] [Interactive SQL]** – connect to the database.
- **DEALLOCATE DESCRIPTOR statement [ESQL]** – reclaim memory for a descriptor.
- **Declaration section [ESQL]** – declare host variables for database communication.
- **DECLARE CURSOR statement [ESQL] [SP]** – declare a cursor.
- **DELETE statement (positioned) [ESQL] [SP]** – delete the row at the current position in a cursor.

- **DESCRIBE statement [ESQL]** – describe the host variables for a particular SQL statement.
- **DISCONNECT statement [ESQL] [Interactive SQL]** – disconnect from database server.
- **DROP STATEMENT statement [ESQL]** – free resources used by a prepared statement.
- **EXECUTE statement [ESQL]** – execute a particular SQL statement.
- **EXPLAIN statement [ESQL]** – explain the optimization strategy for a particular cursor.
- **FETCH statement [ESQL] [SP]** – fetch a row from a cursor.
- **GET DATA statement [ESQL]** – fetch long values from a cursor.
- **GET DESCRIPTOR statement [ESQL]** – retrieve information about a variable in a SQLDA.
- **GET OPTION statement [ESQL]** – get the setting for a particular database option.
- **INCLUDE statement [ESQL]** – include a file for SQL preprocessing.
- **OPEN statement [ESQL] [SP]** – open a cursor.
- **PREPARE statement [ESQL]** – prepare a particular SQL statement.
- **PUT statement [ESQL]** – insert a row into a cursor.
- **SET CONNECTION statement [Interactive SQL] [ESQL]** – change active connection.
- **SET DESCRIPTOR statement [ESQL]** – describe the variables in a SQLDA and place data into the SQLDA.
- **SET SQLCA statement [ESQL]** – use a SQLCA other than the default global one.
- **UPDATE (positioned) statement [ESQL] [SP]** – update the row at the current location of a cursor.
- **WHENEVER statement [ESQL]** – specify actions to occur on errors in SQL statements.

SAP Sybase IQ Database API for C/C++

The SAP Sybase IQ C application programming interface (API) is a data access API for the C/C++ languages. The C API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using the SAP Sybase IQ C API, your C/C++ applications have direct access to SAP Sybase IQ database servers.

sqlany_affected_rows(a_sqlany_stmt *) method

Returns the number of rows affected by execution of the prepared statement.

Syntax

```
public sacapi_i32 sqlany_affected_rows ( a_sqlany_stmt *
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement that was prepared and executed successfully with no result set returned. For example, an INSERT, UPDATE or DELETE statement was executed.

Returns

The number of rows affected or -1 on failure.

sqlany_bind_param(a_sqlany_stmt *, sacapi_u32 , a_sqlany_bind_param *) method

Bind a user-supplied buffer as a parameter to the prepared statement.

Syntax

```
public sacapi_bool sqlany_bind_param ( a_sqlany_stmt *
sqlany_stmt, sacapi_u32 index, a_sqlany_bind_param * param)
```

Parameters

- **sqlany_stmt** – A statement prepared successfully using sqlany_prepare().
- **index** – The index of the parameter. This number must be between 0 and sqlany_num_params() - 1.
- **param** – A a_sqlany_bind_param structure description of the parameter to be bound.

Returns

1 on success or 0 on unsuccessful.

sqlany_cancel(a_sqlany_connection *) method

Cancel an outstanding request on a connection.

Syntax

```
public void sqlany_cancel ( a_sqlany_connection * sqlany_conn)
```

Parameters

- **sqlany_conn** – A connection object with a connection established using `sqlany_connect()`.

sqlany_clear_error(a_sqlany_connection *) method

Clears the last stored error code.

Syntax

```
public void sqlany_clear_error ( a_sqlany_connection *  
sqlany_conn)
```

Parameters

- **sqlany_conn** – A connection object returned from `sqlany_new_connection()`.

sqlany_client_version(char *, size_t) method

Returns the current client version.

Syntax

```
public sacapi_bool sqlany_client_version (char * buffer, size_t  
len)
```

Parameters

- **buffer** – The buffer to be filled with the client version string.
- **len** – The length of the buffer supplied.

Returns

1 when successful or 0 when unsuccessful.

Usage

This method fills the buffer passed with the major, minor, patch, and build number of the client library. The buffer will be null-terminated.

sqlany_client_version_ex(a_sqlany_interface_context *, char *, size_t) method

Returns the current client version.

Syntax

```
public sacapi_bool sqlany_client_version_ex
( a_sqlany_interface_context * context, char * buffer, size_t
len)
```

Parameters

- **context** – object that was create with sqlany_init_ex()
- **buffer** – The buffer to be filled with the client version string.
- **len** – The length of the buffer supplied.

Returns

1 when successful or 0 when unsuccessful.

Usage

This method fills the buffer passed with the major, minor, patch, and build number of the client library. The buffer will be null-terminated.

sqlany_commit(a_sqlany_connection *) method

Commits the current transaction.

Syntax

```
public sacapi_bool sqlany_commit ( a_sqlany_connection *
sqlany_conn)
```

Parameters

- **sqlany_conn** – The connection object on which the commit operation is performed.

Returns

1 when successful or 0 when unsuccessful.

sqlany_connect(a_sqlany_connection *, const char *) method

Creates a connection to a SQL Anywhere database server using the supplied connection object and connection string.

Syntax

```
public sacapi_bool sqlany_connect ( a_sqlany_connection *
sqlany_conn, const char * str)
```

Parameters

- **sqlany_conn** – A connection object created by `sqlany_new_connection()`.
- **str** – A SQL Anywhere connection string.

Returns

1 if the connection is established successfully or 0 when the connection fails. Use `sqlany_error()` to retrieve the error code and message.

Usage

The supplied connection object must first be allocated using `sqlany_new_connection()`.

The following example demonstrates how to retrieve the error code of a failed connection attempt:

```
a_sqlany_connection * sqlany_conn;
sqlany_conn = sqlany_new_connection();
if( !sqlany_connect( sqlany_conn, "uid=dba;pwd=sql" ) ) {
    char reason[SACAPI_ERROR_SIZE];
    sacapi_i32 code;
    code = sqlany_error( sqlany_conn, reason, sizeof(reason) );
    printf( "Connection failed. Code: %d Reason: %s\n", code,
reason );
} else {
    printf( "Connected successfully!\n" );
    sqlany_disconnect( sqlany_conn );
}
sqlany_free_connection( sqlany_conn );
```

For more information on connecting to a SQL Anywhere database server, see Connection parameters and SQL Anywhere database connections.

sqlany_describe_bind_param(a_sqlany_stmt *, sacapi_u32 , a_sqlany_bind_param *) method

Describes the bind parameters of a prepared statement.

Syntax

```
public sacapi_bool sqlany_describe_bind_param ( a_sqlany_stmt
* sqlany_stmt, sacapi_u32 index, a_sqlany_bind_param * param)
```

Parameters

- **sqlany_stmt** – A statement prepared successfully using `sqlany_prepare()`.
- **index** – The index of the parameter. This number must be between 0 and `sqlany_num_params() - 1`.
- **param** – A `a_sqlany_bind_param` structure that is populated with information.

Returns

1 when successful or 0 when unsuccessful.

Usage

This function allows the caller to determine information about prepared statement parameters. The type of prepared statement, stored procedured or a DML, determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

sqlany_disconnect(a_sqlany_connection *) method

Disconnects an already established SQL Anywhere connection.

Syntax

```
public sacapi_bool sqlany_disconnect ( a_sqlany_connection *
sqlany_conn)
```

Parameters

- **sqlany_conn** – A connection object with a connection established using `sqlany_connect()`.

Returns

1 when successful or 0 when unsuccessful.

Usage

All uncommitted transactions are rolled back.

sqlany_error(a_sqlany_connection *, char *, size_t) method

Retrieves the last error code and message stored in the connection object.

Syntax

```
public sacapi_i32 sqlany_error ( a_sqlany_connection *  
sqlany_conn, char * buffer, size_t size)
```

Parameters

- **sqlany_conn** – A connection object returned from `sqlany_new_connection()`.
- **buffer** – A buffer to be filled with the error message.
- **size** – The size of the supplied buffer.

Returns

The last error code. Positive values are warnings, negative values are errors, and 0 indicates success.

Usage

For more information on SQLCODE error messages, see SQL Anywhere error messages sorted by SQLCODE.

sqlany_execute(a_sqlany_stmt *) method

Executes a prepared statement.

Syntax

```
public sacapi_bool sqlany_execute ( a_sqlany_stmt *  
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement prepared successfully using `sqlany_prepare()`.

Returns

1 if the statement is executed successfully or 0 on failure.

Usage

You can use `sqlany_num_cols()` to verify if the executed statement returned a result set.

The following example shows how to execute a statement that does not return a result set:

```

a_sqlany_stmt *      stmt;
int                 i;
a_sqlany_bind_param param;

                                stmt = sqlany_prepare( sqlany_conn, "insert into
moe(id,value) values( ?,? )" );
if( stmt ) {
    sqlany_describe_bind_param( stmt, 0, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
    sqlany_bind_param( stmt, 0, &param );

                                sqlany_describe_bind_param( stmt, 1, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
    sqlany_bind_param( stmt, 1, &param );

                                for( i = 0; i < 10; i++ ) {
        if( !sqlany_execute( stmt ) ) {
            // call sqlany_error()
        }
    }
    sqlany_free_stmt( stmt );
}

```

sqlany_execute_direct(a_sqlany_connection *, const char *) method

Executes the SQL statement specified by the string argument and possibly returns a result set.

Syntax

```

public a_sqlany_stmt * sqlany_execute_direct
( a_sqlany_connection * sqlany_conn, const char * sql_str)

```

Parameters

- **sqlany_conn** – A connection object with a connection established using `sqlany_connect()`.
- **sql_str** – A SQL string. The SQL string should not have parameters such as `?`.

Returns

A statement handle if the function executes successfully, NULL when the function executes unsuccessfully.

Usage

Use this method if you want to prepare and execute a statement, or instead of calling `sqlany_prepare()` followed by `sqlany_execute()`.

The following example shows how to execute a statement that returns a result set:

```
a_sqlany_stmt * stmt;

        stmt = sqlany_execute_direct( sqlany_conn, "select *
from employees" );
if( stmt && sqlany_num_cols( stmt ) > 0 ) {
    while( sqlany_fetch_next( stmt ) ) {
        int i;
        for( i = 0; i < sqlany_num_cols( stmt ); i++ ) {
            // Get column i data
        }
    }
    sqlany_free_stmt( stmt );
}
```

Note: This function cannot be used for executing a SQL statement with parameters.

sqlany_execute_immediate(a_sqlany_connection *, const char *) method

Executes the supplied SQL statement immediately without returning a result set.

Syntax

```
public sacapi_bool sqlany_execute_immediate
( a_sqlany_connection * sqlany_conn, const char * sql)
```


Parameters

- **sqlany_conn** – A connection object with a connection established using `sqlany_connect()`.
- **sql** – A string representing the SQL statement to be executed.

Returns

1 on success or 0 on failure.

Usage

This function is useful for SQL statements that do not return a result set.

sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) method

Moves the current row in the result set to the row number specified and then fetches the data at that row.

Syntax

```
public sacapi_bool sqlany_fetch_absolute ( a_sqlany_stmt *
sqlany_stmt, sacapi_i32 row_num)
```

Parameters

- **sqlany_stmt** – A statement object that was executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **row_num** – The row number to be fetched. The first row is 1, the last row is -1.

Returns

1 if the fetch was successfully, 0 when the fetch is unsuccessful.

sqlany_fetch_next(a_sqlany_stmt *) method

Returns the next row from the result set.

Syntax

```
public sacapi_bool sqlany_fetch_next ( a_sqlany_stmt *
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement object that was executed by `sqlany_execute()` or `sqlany_execute_direct()`.

Returns

1 if the fetch was successfully, 0 when the fetch is unsuccessful.

Usage

This function fetches the next row from the result set. When the result object is first created, the current row pointer is set to before the first row, that is, row 0. This function first advances the row pointer and then fetches the data at the new row.

sqlany_finalize_interface(SQLAnywhereInterface *) method

Unloads the C API DLL library and resets the SQLAnywhereInterface structure.

Syntax

```
public void sqlany_finalize_interface ( SQLAnywhereInterface *  
api)
```

Parameters

- **api** – An initialized structure to finalize.

Usage

Use the following statement to include the function prototype:

```
#include "sacapidll.h"
```

Use this method to finalize and free resources associated with the SQL Anywhere C API DLL.

Examples of how the `sqlany_finalize_interface` method is used can be found in the C API examples in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

sqlany_fini() method

Finalizes the interface.

Syntax

```
public void sqlany_fini ()
```

Usage

Frees any resources allocated by the API.

sqlany_fini_ex(a_sqlany_interface_context *) method

Finalize the interface that was created using the specified context.

Syntax

```
public void sqlany_fini_ex ( a_sqlany_interface_context *  
context)
```

Parameters

- **context** – A context object that was returned from `sqlany_init_ex()`

sqlany_free_connection(a_sqlany_connection *) method

Frees the resources associated with a connection object.

Syntax

```
public void sqlany_free_connection ( a_sqlany_connection *  
sqlany_conn)
```

Parameters

- **sqlany_conn** – A connection object created with `sqlany_new_connection()`.

sqlany_free_stmt(a_sqlany_stmt *) method

Frees resources associated with a prepared statement object.

Syntax

```
public void sqlany_free_stmt ( a_sqlany_stmt * sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement object returned by the successful execution of `sqlany_prepare()` or `sqlany_execute_direct()`.

sqlany_get_bind_param_info(a_sqlany_stmt *, sacapi_u32 , a_sqlany_bind_param_info *) method

Retrieves information about the parameters that were bound using `sqlany_bind_param()`.

Syntax

```
public sacapi_bool sqlany_get_bind_param_info ( a_sqlany_stmt  
* sqlany_stmt, sacapi_u32 index, a_sqlany_bind_param_info * info)
```

Parameters

- **sqlany_stmt** – A statement prepared successfully using `sqlany_prepare()`.
- **index** – The index of the parameter. This number should be between 0 and `sqlany_num_params() - 1`.
- **info** – A `sqlany_bind_param_info` buffer to be populated with the bound parameter's information.

Returns

1 on success or 0 on failure.

sqlany_get_column(a_sqlany_stmt *, sacapi_u32 , a_sqlany_data_value *) method

Fills the supplied buffer with the value fetched for the specified column.

Syntax

```
public sacapi_bool sqlany_get_column ( a_sqlany_stmt *
sqlany_stmt, sacapi_u32 col_index, a_sqlany_data_value * buffer)
```

Parameters

- **sqlany_stmt** – A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **col_index** – The number of the column to be retrieved. A column number is between 0 and `sqlany_num_cols() - 1`.
- **buffer** – A `a_sqlany_data_value` object to be filled with the data fetched for column `col_index`.

Returns

1 on success or 0 for failure. A failure can happen if any of the parameters are invalid or if there is not enough memory to retrieve the full value from the SQL Anywhere database server.

Usage

For `A_BINARY` and `A_STRING *` data types, `value->buffer` points to an internal buffer associated with the result set. Do not rely upon or alter the content of the pointer buffer as it changes when a new row is fetched or when the result set object is freed. Users should copy the data out of those pointers into their own buffers.

The `value->length` field indicates the number of valid characters that `value->buffer` points to. The data returned in `value->buffer` is not null-terminated. This function fetches all the returned values from the SQL Anywhere database server. For example, if the column contains a blob, this function attempts to allocate enough memory to hold that value. If you do not want to allocate memory, use `sqlany_get_data()` instead.

sqlany_get_column_info(a_sqlany_stmt *, sacapi_u32 , a_sqlany_column_info *) method

Retrieves column metadata information and fills the a_sqlany_column_info structure with information about the column.

Syntax

```
public sacapi_bool sqlany_get_column_info ( a_sqlany_stmt *  
sqlany_stmt, sacapi_u32 col_index, a_sqlany_column_info * buffer)
```

Parameters

- **sqlany_stmt** – A statement object created by sqlany_prepare() or sqlany_execute_direct().
- **col_index** – The column number between 0 and sqlany_num_cols() - 1.
- **buffer** – A column info structure to be filled with column information.

Returns

1 on success or 0 if the column index is out of range, or if the statement does not return a result set.

sqlany_get_data(a_sqlany_stmt *, sacapi_u32 , size_t, void *, size_t) method

Retrieves the data fetched for the specified column into the supplied buffer memory.

Syntax

```
public sacapi_i32 sqlany_get_data ( a_sqlany_stmt * sqlany_stmt,  
sacapi_u32 col_index, size_t offset, void * buffer, size_t size)
```

Parameters

- **sqlany_stmt** – A statement object executed by sqlany_execute() or sqlany_execute_direct().
- **col_index** – The number of the column to be retrieved. A column number is between 0 and sqlany_num_cols() - 1.
- **offset** – The starting offset of the data to get.
- **buffer** – A buffer to be filled with the contents of the column. The buffer pointer must be aligned correctly for the data type copied into it.

- **size** – The size of the buffer in bytes. The function fails if you specify a size greater than $2^{31} - 1$.

Returns

The number of bytes successfully copied into the supplied buffer. This number must not exceed $2^{31} - 1$. 0 indicates that no data remains to be copied. -1 indicates a failure.

sqlany_get_data_info(a_sqlany_stmt *, sacapi_u32 , a_sqlany_data_info *) method

Retrieves information about the data that was fetched by the last fetch operation.

Syntax

```
public sacapi_bool sqlany_get_data_info ( a_sqlany_stmt *
sqlany_stmt, sacapi_u32 col_index, a_sqlany_data_info * buffer)
```

Parameters

- **sqlany_stmt** – A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **col_index** – The column number between 0 and `sqlany_num_cols() - 1`.
- **buffer** – A data info buffer to be filled with the metadata about the data fetched.

Returns

1 on success, and 0 on failure. Failure is returned when any of the supplied parameters are invalid.

sqlany_get_next_result(a_sqlany_stmt *) method

Advances to the next result set in a multiple result set query.

Syntax

```
public sacapi_bool sqlany_get_next_result ( a_sqlany_stmt *
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.

Returns

1 if the statement successfully advances to the next result set, 0 otherwise.

Usage

If a query (such as a call to a stored procedure) returns multiple result sets, then this function advances from the current result set to the next.

The following example demonstrates how to advance to the next result set in a multiple result set query:

```
stmt = sqlany_execute_direct( sqlany_conn, "call
my_multiple_results_procedure()" );
if( result ) {
    do {
        while( sqlany_fetch_next( stmt ) ) {
            // get column data
        }
    } while( sqlany_get_next_result( stmt ) );
    sqlany_free_stmt( stmt );
}
```

sqlany_init(const char *, sacapi_u32 , sacapi_u32 *) method

Initializes the interface.

Syntax

```
public sacapi_bool sqlany_init (const char * app_name,
sacapi_u32 api_version, sacapi_u32 * version_available)
```

Parameters

- **app_name** – A string that names the application that is using the API. For example, "PHP", "PERL", or "RUBY".
- **api_version** – The version of the compiled application.
- **version_available** – An optional argument to return the maximum supported API version.

Returns

1 on success, 0 otherwise

Usage

The following example demonstrates how to initialize the SQL Anywhere C API DLL:

```
sacapi_u32 api_version;
if( sqlany_init( "PHP", SQLANY_API_VERSION_1, &api_version ) ) {
    printf( "Interface initialized successfully!\n" );
}
```



```

} else {
    printf( "Failed to initialize the interface! Supported version=
%d\n", api_version );
}

```

sqlany_init_ex(const char *, sacapi_u32 , sacapi_u32 *) method

Initializes the interface using a context.

Syntax

```

public a_sqlany_interface_context * sqlany_init_ex (const char
* app_name, sacapi_u32 api_version, sacapi_u32 * version_available)

```

Parameters

- **app_name** – A string that names the API used, for example "PHP", "PERL", or "RUBY".
- **api_version** – The current API version that the application is using. This should normally be one of the SQLANY_API_VERSION_* macros
- **version_available** – An optional argument to return the maximum API version that is supported.

Returns

a context object on success and NULL on failure.

sqlany_initialize_interface(SQLAnywhereInterface *, const char *) method

Initializes the SQLAnywhereInterface object and loads the DLL dynamically.

Syntax

```

public int sqlany_initialize_interface ( SQLAnywhereInterface
* api, const char * optional_path_to_dll)

```

Parameters

- **api** – An API structure to initialize.
- **optional_path_to_dll** – An optional argument that specifies a path to the SQL Anywhere C API DLL.

Returns

1 on successful initialization, and 0 on failure.

Usage

Use the following statement to include the function prototype:

```
#include "sacapidll.h"
```

This function attempts to load the SQL Anywhere C API DLL dynamically and looks up all the entry points of the DLL. The fields in the `SQLAnywhereInterface` structure are populated to point to the corresponding functions in the DLL. If the optional path argument is `NULL`, the environment variable `SQLANY_DLL_PATH` is checked. If the variable is set, the library attempts to load the DLL specified by the environment variable. If that fails, the interface attempts to load the DLL directly (this relies on the environment being setup correctly).

Examples of how the `sqlany_initialize_interface` method is used can be found in the C API examples in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

sqlany_make_connection(void *) method

Creates a connection object based on a supplied DBLIB SQLCA pointer.

Syntax

```
public a_sqlany_connection * sqlany_make_connection (void *  
arg)
```

Parameters

- **arg** – A void * pointer to a DBLIB SQLCA object.

Returns

A connection object.

sqlany_make_connection_ex(a_sqlany_interface_context *, void *) method

Creates a connection object based on a supplied DBLIB SQLCA pointer and context.

Syntax

```
public a_sqlany_connection * sqlany_make_connection_ex  
( a_sqlany_interface_context * context, void * arg)
```

Parameters

- **context** – A valid context object that was created by `sqlany_init_ex()`

- **arg** – A void * pointer to a DBLIB SQLCA object.

Returns

A connection object.

sqlany_new_connection(void) method

Creates a connection object.

Syntax

```
public a_sqlany_connection * sqlany_new_connection (void )
```

Returns

A connection object

Usage

You must create an API connection object before establishing a database connection. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time. In addition, not more than one thread is allowed to access a connection object at a time. Undefined behavior or a failure occurs when multiple threads attempt to access a connection object simultaneously.

sqlany_new_connection_ex(a_sqlany_interface_context *) method

Creates a connection object using a context.

Syntax

```
public a_sqlany_connection * sqlany_new_connection_ex  
( a_sqlany_interface_context * context)
```

Parameters

- **context** – A context object that was returned from sqlany_init_ex()

Returns

A connection object

sqlany_num_cols(a_sqlany_stmt *) method

Returns number of columns in the result set.

Syntax

```
public sacapi_i32 sqlany_num_cols ( a_sqlany_stmt *  
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement object created by `sqlany_prepare()` or `sqlany_execute_direct()`.

Returns

The number of columns in the result set or -1 on a failure.

sqlany_num_params(a_sqlany_stmt *) method

Returns the number of parameters expected for a prepared statement.

Syntax

```
public sacapi_i32 sqlany_num_params ( a_sqlany_stmt *  
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement object returned by the successful execution of `sqlany_prepare()`.

Returns

The expected number of parameters, or -1 if the statement object is not valid.

sqlany_num_rows(a_sqlany_stmt *) method

Returns the number of rows in the result set.

Syntax

```
public sacapi_i32 sqlany_num_rows ( a_sqlany_stmt *  
sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement object that was executed by `sqlany_execute()` or `sqlany_execute_direct()`.

Returns

The number rows in the result set. If the number of rows is an estimate, the number returned is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

Usage

By default this function only returns an estimate. To return an exact count, set the `row_counts` option on the connection. For more information on the `row_counts` option, see `row_counts` option [database].

sqlany_prepare(a_sqlany_connection *, const char *) method

Prepares a supplied SQL string.

Syntax

```
public a_sqlany_stmt * sqlany_prepare ( a_sqlany_connection *
sqlany_conn, const char * sql_str)
```

Parameters

- **sqlany_conn** – A connection object with a connection established using `sqlany_connect()`.
- **sql_str** – The SQL statement to be prepared.

Returns

A handle to a SQL Anywhere statement object. The statement object can be used by `sqlany_execute()` to execute the statement.

Usage

Execution does not happen until `sqlany_execute()` is called. The returned statement object should be freed using `sqlany_free_stmt()`.

The following statement demonstrates how to prepare a SELECT SQL string:

```
char * str;
a_sqlany_stmt * stmt;
```

```
        str = "select * from employees where salary >= ?";
stmt = sqlany_prepare( sqlany_conn, str );
if( stmt == NULL ) {
    // Failed to prepare statement, call sqlany_error() for more info
}
```

sqlany_reset(a_sqlany_stmt *) method

Resets a statement to its prepared state condition.

Syntax

```
public sacapi_bool sqlany_reset ( a_sqlany_stmt * sqlany_stmt)
```

Parameters

- **sqlany_stmt** – A statement prepared successfully using `sqlany_prepare()`.

Returns

1 on success, 0 on failure.

sqlany_rollback(a_sqlany_connection *) method

Rolls back the current transaction.

Syntax

```
public sacapi_bool sqlany_rollback ( a_sqlany_connection *
sqlany_conn)
```

Parameters

- **sqlany_conn** – The connection object on which the rollback operation is to be performed.

Returns

1 on success, 0 otherwise.

sqlany_send_param_data(a_sqlany_stmt *, sacapi_u32 , char *, size_t) method

Sends data as part of a bound parameter.

Syntax

```
public sacapi_bool sqlany_send_param_data ( a_sqlany_stmt *
sqlany_stmt, sacapi_u32 index, char * buffer, size_t size)
```

Parameters

- **sqlany_stmt** – A statement prepared successfully using `sqlany_prepare()`.
- **index** – The index of the parameter. This should be a number between 0 and `sqlany_num_params() - 1`.
- **buffer** – The data to be sent.
- **size** – The number of bytes to send.

Returns

1 on success or 0 on failure.

Usage

This method can be used to send a large amount of data for a bound parameter in chunks.

sqlany_sqlstate(a_sqlany_connection *, char *, size_t) method

Retrieves the current SQLSTATE.

Syntax

```
public size_t sqlany_sqlstate ( a_sqlany_connection *
sqlany_conn, char * buffer, size_t size)
```

Parameters

- **sqlany_conn** – A connection object returned from `sqlany_new_connection()`.
- **buffer** – A buffer to be filled with the current 5-character SQLSTATE.
- **size** – The buffer size.

Returns

The number of bytes copied into the buffer.

Usage

For more information on SQLSTATE error messages, see SQL Anywhere error messages sorted by SQLSTATE.

a_sqlany_data_direction() enumeration

A data direction enumeration.

Enum Constant Summary

- **DD_INVALID** – Invalid data direction.
- **DD_INPUT** – Input-only host variables.
- **DD_OUTPUT** – Output-only host variables.
- **DD_INPUT_OUTPUT** – Input and output host variables.

a_sqlany_data_type() enumeration

Specifies the data type being passed in or retrieved.

Enum Constant Summary

- **A_INVALID_TYPE** – Invalid data type.
- **A_BINARY** – Binary data. Binary data is treated as-is and no character set conversion is performed.
- **A_STRING** – String data. The data where character set conversion is performed.
- **A_DOUBLE** – Double data. Includes float values.
- **A_VAL64** – 64-bit integer.
- **A_UVAL64** – 64-bit unsigned integer.
- **A_VAL32** – 32-bit integer.
- **A_UVAL32** – 32-bit unsigned integer.
- **A_VAL16** – 16-bit integer.
- **A_UVAL16** – 16-bit unsigned integer.
- **A_VAL8** – 8-bit integer.
- **A_UVAL8** – 8-bit unsigned integer.

a_sqlany_native_type() enumeration

An enumeration of the native types of values as described by the server.

Enum Constant Summary

- **DT_NOTYPE** – No data type.
- **DT_DATE** – Null-terminated character string that is a valid date.
- **DT_TIME** – Null-terminated character string that is a valid time.
- **DT_TIMESTAMP** – Null-terminated character string that is a valid timestamp.
- **DT_VARCHAR** – Varying length character string, in the CHAR character set, with a two-byte length field. The maximum length is 32765 bytes. When sending data, you must set the length field. When fetching data, the database server sets the length field. The data is not null-terminated or blank-padded.
- **DT_FIXCHAR** – Fixed-length blank-padded character string, in the CHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.
- **DT_LONGVARCHAR** – Long varying length character string, in the CHAR character set.
- **DT_STRING** – Null-terminated character string, in the CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.
- **DT_DOUBLE** – 8-byte floating-point number.
- **DT_FLOAT** – 4-byte floating-point number.
- **DT_DECIMAL** – Packed decimal number (proprietary format).
- **DT_INT** – 32-bit signed integer.
- **DT_SMALLINT** – 16-bit signed integer.
- **DT_BINARY** – Varying length binary data with a two-byte length field. The maximum length is 32765 bytes. When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field.
- **DT_LONGBINARY** – Long binary data.
- **DT_TINYINT** – 8-bit signed integer.
- **DT_BIGINT** – 64-bit signed integer.
- **DT_UNSYNINT** – 32-bit unsigned integer.
- **DT_UNSSMALLINT** – 16-bit unsigned integer.
- **DT_UNSBIGINT** – 64-bit unsigned integer.
- **DT_BIT** – 8-bit signed integer.
- **DT_LONGNVARCHAR** – Long varying length character string, in the NCHAR character set.

SACAPI_ERROR_SIZE variable

Returns the minimal error buffer size.

Syntax

```
#define SACAPI_ERROR_SIZE
```

SQLANY_API_VERSION_1 variable

Defines to indicate the API versions.

Syntax

```
#define SQLANY_API_VERSION_1
```

SQLANY_API_VERSION_2 variable

Version 2 introduced the "_ex" functions and the ability to cancel requests.

Syntax

```
#define SQLANY_API_VERSION_2
```

SQLAnywhereInterface structure

The SQL Anywhere C API interface structure.

Syntax

```
typedef struct SQLAnywhereInterface
```

Remarks

Only one instance of this structure is required in your application environment. This structure is initialized by the `sqlany_initialize_interface` method. It attempts to load the SQL Anywhere C API DLL or shared object dynamically and looks up all the entry points of the DLL. The fields in the `SQLAnywhereInterface` structure is populated to point to the corresponding functions in the DLL.

dll_handle void *

DLL handle.

Syntax

```
public void * dll_handle;
```

initialized int

Flag to know if initialized or not.

Syntax

```
public int initialized;
```

sqlany_affected_rows void *

Pointer to sqlany_affected_rows() function.

Syntax

```
public void * sqlany_affected_rows;
```

sqlany_bind_param void *

Pointer to sqlany_bind_param() function.

Syntax

```
public void * sqlany_bind_param;
```

sqlany_cancel void *

Pointer to sqlany_cancel() function.

Syntax

```
public void * sqlany_cancel;
```

sqlany_clear_error void *

Pointer to sqlany_clear_error() function.

Syntax

```
public void * sqlany_clear_error;
```

sqlany_client_version void *

Pointer to sqlany_client_version() function.

Syntax

```
public void * sqlany_client_version;
```

sqlany_client_version_ex void *

Pointer to sqlany_client_version_ex() function.

Syntax

```
public void * sqlany_client_version_ex;
```

sqlany_commit void *

Pointer to sqlany_commit() function.

Syntax

```
public void * sqlany_commit;
```

sqlany_connect void *

Pointer to sqlany_connect() function.

Syntax

```
public void * sqlany_connect;
```

sqlany_describe_bind_param void *

Pointer to sqlany_describe_bind_param() function.

Syntax

```
public void * sqlany_describe_bind_param;
```

sqlany_disconnect void *

Pointer to sqlany_disconnect() function.

Syntax

```
public void * sqlany_disconnect;
```

sqlany_error void *

Pointer to sqlany_error() function.

Syntax

```
public void * sqlany_error;
```

sqlany_execute void *

Pointer to sqlany_execute() function.

Syntax

```
public void * sqlany_execute;
```

sqlany_execute_direct void *

Pointer to sqlany_execute_direct() function.

Syntax

```
public void * sqlany_execute_direct;
```

sqlany_execute_immediate void *

Pointer to sqlany_execute_immediate() function.

Syntax

```
public void * sqlany_execute_immediate;
```

sqlany_fetch_absolute void *

Pointer to sqlany_fetch_absolute() function.

Syntax

```
public void * sqlany_fetch_absolute;
```

sqlany_fetch_next void *

Pointer to sqlany_fetch_next() function.

Syntax

```
public void * sqlany_fetch_next;
```

sqlany_fini void *

Pointer to `sqlany_fini()` function.

Syntax

```
public void * sqlany_fini;
```

sqlany_fini_ex void *

Pointer to `sqlany_fini_ex()` function.

Syntax

```
public void * sqlany_fini_ex;
```

sqlany_free_connection void *

Pointer to `sqlany_free_connection()` function.

Syntax

```
public void * sqlany_free_connection;
```

sqlany_free_stmt void *

Pointer to `sqlany_free_stmt()` function.

Syntax

```
public void * sqlany_free_stmt;
```

sqlany_get_bind_param_info void *

Pointer to `sqlany_get_bind_param_info()` function.

Syntax

```
public void * sqlany_get_bind_param_info;
```

sqlany_get_column void *

Pointer to `sqlany_get_column()` function.

Syntax

```
public void * sqlany_get_column;
```

sqlany_get_column_info void *

Pointer to `sqlany_get_column_info()` function.

Syntax

```
public void * sqlany_get_column_info;
```

sqlany_get_data void *

Pointer to `sqlany_get_data()` function.

Syntax

```
public void * sqlany_get_data;
```

sqlany_get_data_info void *

Pointer to `sqlany_get_data_info()` function.

Syntax

```
public void * sqlany_get_data_info;
```

sqlany_get_next_result void *

Pointer to `sqlany_get_next_result()` function.

Syntax

```
public void * sqlany_get_next_result;
```

sqlany_init void *

Pointer to `sqlany_init()` function.

Syntax

```
public void * sqlany_init;
```

sqlany_init_ex void *

Pointer to `sqlany_init_ex()` function.

Syntax

```
public void * sqlany_init_ex;
```

sqlany_make_connection void *

Pointer to `sqlany_make_connection()` function.

Syntax

```
public void * sqlany_make_connection;
```

sqlany_make_connection_ex void *

Pointer to `sqlany_make_connection_ex()` function.

Syntax

```
public void * sqlany_make_connection_ex;
```

sqlany_new_connection void *

Pointer to `sqlany_new_connection()` function.

Syntax

```
public void * sqlany_new_connection;
```

sqlany_new_connection_ex void *

Pointer to `sqlany_new_connection_ex()` function.

Syntax

```
public void * sqlany_new_connection_ex;
```

sqlany_num_cols void *

Pointer to `sqlany_num_cols()` function.

Syntax

```
public void * sqlany_num_cols;
```

sqlany_num_params void *

Pointer to `sqlany_num_params()` function.

Syntax

```
public void * sqlany_num_params;
```


sqlany_num_rows void *

Pointer to sqlany_num_rows() function.

Syntax

```
public void * sqlany_num_rows;
```

sqlany_prepare void *

Pointer to sqlany_prepare() function.

Syntax

```
public void * sqlany_prepare;
```

sqlany_reset void *

Pointer to sqlany_reset() function.

Syntax

```
public void * sqlany_reset;
```

sqlany_rollback void *

Pointer to sqlany_rollback() function.

Syntax

```
public void * sqlany_rollback;
```

sqlany_send_param_data void *

Pointer to sqlany_send_param_data() function.

Syntax

```
public void * sqlany_send_param_data;
```

sqlany_sqlstate void *

Pointer to sqlany_sqlstate() function.

Syntax

```
public void * sqlany_sqlstate;
```

a_sqlany_bind_param structure

A bind parameter structure used to bind parameter and prepared statements.

Syntax

```
typedef struct a_sqlany_bind_param
```

Remarks

To view examples of the a_sqlany_bind_param structure in use, see any of the following sample files in the sdk\dbcapi\examples directory of your SQL Anywhere installation.

- preparing_statements.cpp
- send_retrieve_full_blob.cpp
- send_retrieve_part_blob.cpp

direction a_sqlany_data_direction

The direction of the data. (input, output, input_output).

Syntax

```
public a_sqlany_data_direction direction;
```

name char *

Name of the bind parameter. This is only used by sqlany_describe_bind_param().

Syntax

```
public char * name;
```

value a_sqlany_data_value

The actual value of the data.

Syntax

```
public a_sqlany_data_value value;
```

a_sqlany_bind_param_info structure

Gets information about the currently bound parameters.

Syntax

```
typedef struct a_sqlany_bind_param_info
```

Remarks

`sqlany_get_bind_param_info()` can be used to populate this structure.

To view examples of the `a_sqlany_bind_param_info` structure in use, see any of the following sample files in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

- `preparing_statements.cpp`
- `send_retrieve_full_blob.cpp`
- `send_retrieve_part_blob.cpp`

direction a_sqlany_data_direction

The direction of the parameter.

Syntax

```
public a_sqlany_data_direction direction;
```

input_value a_sqlany_data_value

Information about the bound input value.

Syntax

```
public a_sqlany_data_value input_value;
```

name char *

A pointer to the name of the parameter.

Syntax

```
public char * name;
```

output_value a_sqlany_data_value

Information about the bound output value.

Syntax

```
public a_sqlany_data_value output_value;
```

a_sqlany_column_info structure

Returns column metadata information.

Syntax

```
typedef struct a_sqlany_column_info
```

Remarks

sqlany_get_column_info() can be used to populate this structure.

To view an example of the a_sqlany_column_info structure in use, see the following sample file in the sdk\dbcapi\examples directory of your SQL Anywhere installation.

- dbcapi_isql.cpp

max_size size_t

The maximum size a data value in this column can take.

Syntax

```
public size_t max_size;
```

name char *

The name of the column (null-terminated).

Syntax

```
public char * name;
```

Remarks

The string can be referenced as long as the result set object is not freed.

native_type a_sqlany_native_type

The native type of the column in the database.

Syntax

```
public a_sqlany_native_type native_type;
```

nullable sacapi_bool

Indicates whether a value in the column can be null.

Syntax

```
public sacapi_bool nullable;
```

precision unsigned short

The precision.

Syntax

```
public unsigned short precision;
```

scale unsigned short

The scale.

Syntax

```
public unsigned short  scale;
```

type a_sqlany_data_type

The column data type.

Syntax

```
public a_sqlany_data_type  type;
```

a_sqlany_data_info structure

Returns metadata information about a column value in a result set.

Syntax

```
typedef struct  a_sqlany_data_info
```

Remarks

`sqlany_get_data_info()` can be used to populate this structure with information about what was last retrieved by a fetch operation.

To view an example of the `a_sqlany_data_info` structure in use, see the following sample file in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

- `send_retrieve_part_blob.cpp`

data_size size_t

The total number of bytes available to be fetched.

Syntax

```
public size_t  data_size;
```

Remarks

This field is only valid after a successful fetch operation.

is_null sacapi_bool

Indicates whether the last fetched data is NULL.

Syntax

```
public sacapi_bool  is_null;
```

Remarks

This field is only valid after a successful fetch operation.

type a_sqlany_data_type

The type of the data in the column.

Syntax

```
public a_sqlany_data_type type;
```

a_sqlany_data_value structure

Returns a description of the attributes of a data value.

Syntax

```
typedef struct a_sqlany_data_value
```

Remarks

To view examples of the `a_sqlany_data_value` structure in use, see any of the following sample files in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

- `dbcapi_isql.cpp`
- `fetching_a_result_set.cpp`
- `send_retrieve_full_blob.cpp`
- `preparing_statements.cpp`

buffer char *

A pointer to user supplied buffer of data.

Syntax

```
public char * buffer;
```

buffer_size size_t

The size of the buffer.

Syntax

```
public size_t buffer_size;
```

is_null sacapi_bool *

A pointer to indicate whether the last fetched data is NULL.

Syntax

```
public sacapi_bool * is_null;
```

length size_t *

A pointer to the number of valid bytes in the buffer. This value must be less than `buffer_size`.

Syntax

```
public size_t * length;
```

type a_sqlany_data_type

The type of the data.

Syntax

```
public a_sqlany_data_type type;
```


Perl DBI Support

DBD::SQLAnywhere is the SAP Sybase IQ database driver for DBI, which is a data access API for the Perl language. The DBI API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using DBI and DBD::SQLAnywhere, your Perl scripts have direct access to SAP Sybase IQ database servers.

DBD::SQLAnywhere

DBD::SQLAnywhere is a driver for the Database Independent Interface for Perl (DBI) module written by Tim Bunce. Once you have installed the DBI module and DBD::SQLAnywhere, you can access and change the information in SAP Sybase IQ databases from Perl.

The DBD::SQLAnywhere driver is thread-safe when using Perl with `ithreads`.

Requirements

The DBD::SQLAnywhere interface requires the following components.

- Perl 5.6.0 or later. On Windows, ActivePerl 5.6.0 build 616 or later is required.
- DBI 1.34 or later.
- A C compiler. On Windows, only the Microsoft Visual C++ compiler is supported.

Installing DBD::SQLAnywhere on Windows

Install the DBD::SQLAnywhere interface on the supported Windows platform to use Perl to access SAP Sybase IQ databases.

Prerequisites

- Make the `iqdemo` database.
- Install ActivePerl 5.6.0 or later. You can use the ActivePerl installer to install Perl and configure your computer. You do not need to recompile Perl.
- Install Microsoft Visual Studio and configure your environment.

If you did not choose to configure your environment at install time, you must set your `PATH`, `LIB`, and `INCLUDE` environment variables correctly before proceeding. Microsoft provides a batch file for this purpose. For 32-bit builds, a batch file called `vcvars32.bat` is included in the `vc\bin` subdirectory of the Visual Studio 2005 or 2008 installation. For 64-bit builds, look for a 64-bit version of this batch file such as

`vcvarsamd64.bat`. Open a new system command prompt and run this batch file before continuing.

For more information about configuring a 64-bit Visual C++ build environment, see <http://msdn.microsoft.com/en-us/library/x4d2c09s.aspx>.

Task

1. At a command prompt, change to the `bin` subdirectory of your ActivePerl installation directory.

The system command prompt is strongly recommended as the following steps may not work from alternative shells.

2. Using the Perl Module Manager, enter the following command.

```
ppm query dbi
```

If `ppm` fails to run, check that Perl is installed correctly.

This command should generate two lines of text similar to those shown below. In this case, the information indicates that ActivePerl version 5.8.1 build 807 is running and that DBI version 1.38 is installed.

```
Querying target 1 (ActivePerl 5.8.1.807)
  1. DBI [1.38] Database independent interface for Perl
```

Later versions of Perl may show instead a table similar to the following. In this case, the information indicates that DBI version 1.58 is installed.

name	version	abstract	area
DBI	1.58	Database independent interface for Perl	perl

If DBI is not installed, you must install it. To do so, enter the following command at the `ppm` prompt.

```
ppm install dbi
```

3. At a command prompt, change to the `SDK\Perl` subdirectory of your SAP Sybase IQ installation.
4. Enter the following commands to build and test `DBD::SQLAnywhere`.

```
perl Makefile.PL
```

```
nmake
```

If for any reason you need to start over, you can run the command `nmake clean` to remove any partially built targets.

5. To test `DBD::SQLAnywhere`, copy the sample database file to your `SDK\Perl` directory and make the tests.

```
copy "%ALLUSERSPROFILE%\SybaseIQ\demo\iqdemo.db" .
```

```
iqsrv16 demo
```

```
nmake test
```

If the tests do not run, ensure that the `bin32` or `bin64` subdirectory of the SAP Sybase IQ installation is in your path.

- To complete the installation, run the following command at the same prompt.

```
nmake install
```

The DBI Perl module and the `DBD::SQLAnywhere` interface are now ready to use.

Installing DBD::SQLAnywhere on Unix

Install the `DBD::SQLAnywhere` interface on the supported Unix platforms to use Perl to access SAP Sybase IQ databases.

Prerequisites

You must have ActivePerl 5.6.0 build 616 or later and a C compiler installed.

Task

- Download the DBI module source from <http://www.cpan.org>.
- Extract the contents of this file into a new directory.
- At a command prompt, change to the new directory and run the following commands to build the DBI module.

```
perl Makefile.PL
```

```
make
```

If for any reason you need to start over, you can use the command `make clean` to remove any partially built targets.

- Use the following command to test the DBI module.

```
make test
```

- To complete the installation, run the following command at the same prompt.

```
make install
```

- Make sure the environment is set up for SAP Sybase IQ.

Depending on which shell you are using, enter the appropriate command to source the SAP Sybase IQ configuration script from the SAP Sybase IQ installation directory:

In this shell...	Use this command...
sh, ksh, or bash	<code>.bin/sa_config.sh</code>

In this shell...	Use this command...
csh or tcsh	source bin/sa_config.csh

- At a shell prompt, change to the `sdk/perl` subdirectory of your SAP Sybase IQ installation.
- At a command prompt, run the following commands to build `DBD::SQLAnywhere`.

```
perl Makefile.PL
```

```
make
```

If for any reason you need to start over, you can use the command `make clean` to remove any partially built targets.

- To test `DBD::SQLAnywhere`, copy the sample database file to your `sdk/perl` directory and make the tests.

```
cp samples-dir/demo.db .
```

```
iqsrv16 demo
```

```
make test
```

If the tests do not run, ensure that the `bin32` or `bin64` subdirectory of the SAP Sybase IQ installation is in your path.

- To complete the installation, run the following command at the same prompt.

```
make install
```

The DBI Perl module and the `DBD::SQLAnywhere` interface are ready for use.

Next

Optionally, you can delete the DBI source tree. It is no longer required.

Perl Scripts That Use `DBD::SQLAnywhere`

This section provides an overview of how to write Perl scripts that use the `DBD::SQLAnywhere` interface.

`DBD::SQLAnywhere` is a driver for the DBI module. Complete documentation for the DBI module is available online at <http://dbi.perl.org>.

The DBI Module

To use the `DBD::SQLAnywhere` interface from a Perl script, you must first tell Perl that you plan to use the DBI module. To do so, include the following line at the top of the file.

```
use DBI;
```

In addition, it is highly recommended that you run Perl in strict mode. This statement, which for example makes explicit variable definitions mandatory, is likely to greatly reduce the

chance that you will run into mysterious errors due to such common mistakes as typographical errors.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
```

The DBI module automatically loads the DBD drivers, including DBD::SQLAnywhere, as required.

How to Open and Close a Database Connection Using Perl DBI

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the connect method. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The parameters to the connect method are as follows:

1. "DBI:SQLAnywhere:" and additional connection parameters separated by semicolons.
2. A user name. Unless this string is blank, ";UID=*value*" is appended to the connection string.
3. A password value. Unless this string is blank, ";PWD=*value*" is appended to the connection string.
4. A pointer to a hash of default values. Settings such as AutoCommit, RaiseError, and PrintError may be set in this manner.

The following code sample opens and closes a connection to the SAP Sybase IQ sample database. You must start the database server and sample database before running this script.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my %defaults = (
    AutoCommit => 1, # Autocommit enabled.
    PrintError  => 0 # Errors not automatically printed.
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$dbh->disconnect;
exit(0);
__END__
```

Optionally, you can append the user name or password value to the data-source string instead of supplying them as separate parameters. If you do so, supply a blank string for the corresponding argument. For example, in the above script may be altered by replacing the statement that opens the connections with these statements:

```

$data_src .= ";UID=$uid";
$data_src .= ";PWD=$pwd";
my $dbh = DBI->connect($data_src, '', '', \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";

```

How to Obtain Result Sets Using Perl DBI

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

SQL statements that return row sets must be prepared before being executed. The prepare method returns a handle to the statement. You use the handle to execute the statement, then retrieve meta information about the result set and the rows of the result set.

```

#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd     = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
              FROM Customers
              ORDER BY GivenName, Surname";

my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);

my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
    my($sel, $dbh) = @_;
    my($row, $sth) = undef;
    $sth = $dbh->prepare($sel);
    $sth->execute;
    print "Fields:      $sth->{NUM_OF_FIELDS}\n";
    print "Params:      $sth->{NUM_OF_PARAMS}\n\n";
    print join("\t\t", @{$sth->{NAME}}), "\n\n";
    while($row = $sth->fetchrow_arrayref) {
        print join("\t\t", @$row), "\n";
    }
    $sth = undef;
}

__END__

```

Prepared statements are not dropped from the database server until the Perl statement handle is destroyed. To destroy a statement handle, reuse the variable or set it to undef. Calling the finish

method does not drop the handle. In fact, the finish method should not be called, except when you have decided not to finish reading a result set.

To detect handle leaks, the SAP Sybase IQ database server limits the number of cursors and prepared statements permitted to a maximum of 50 per connection by default. The resource governor automatically generates an error if these limits are exceeded. If you get this error, check for undestroyed statement handles. Use `prepare_cached` sparingly, as the statement handles are not destroyed.

If necessary, you can alter these limits by setting the `max_cursor_count` and `max_statement_count` options.

How to Process Multiple Result Sets Using Perl DBI

The method for handling multiple result sets from a query involves wrapping the fetch loop within another loop that moves between result sets.

SQL statements that return multiple result sets must be prepared before being executed. The `prepare` method returns a handle to the statement. You use the handle to execute the statement, then retrieve meta information about the result set and the rows of each of the result sets.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd     = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
               FROM Customers
               ORDER BY GivenName, Surname;
               SELECT *
               FROM Departments
               ORDER BY DepartmentID";

my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
    my($sel, $dbh) = @_;
    my($row, $sth) = undef;
    $sth = $dbh->prepare($sel);
    $sth->execute;
    do {
        print "Fields:      $sth->{NUM_OF_FIELDS}\n";
        print "Params:      $sth->{NUM_OF_PARAMS}\n\n";
    }
}
```

```

        print join("\t\t", @{$sth->{NAME}}), "\n\n";
        while($row = $sth->fetchrow_arrayref) {
            print join("\t\t", @$row), "\n";
        }
        print "---end of results---\n\n";
    } while (defined $sth->more_results);
    $sth = undef;
}
END_

```

How to Insert Rows Using Perl DBI

Inserting rows requires a handle to an open connection. The simplest method is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The statement is first prepared, and then executed once per new row. The new row values are supplied as parameters to the execute method.

The following sample program inserts two new customers. Although the row values appear as literal strings, you may want to read the values from a file.

```

#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd     = "sql";
my $ins_stmt = "INSERT INTO Customers (ID, GivenName, Surname,
        Street, City, State, Country, PostalCode,
        Phone, CompanyName)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Can't connect to $data_src: $DBI::errstr\n";
$db_insert($ins_stmt, $dbh);
$dbh->commit;
$dbh->disconnect;
exit(0);

sub db_insert {
    my($ins, $dbh) = @_;
    my($sth) = undef;
    my @rows = (
        "801,Alex,Alt,5 Blue Ave,New York,NY,USA,
10012,5185553434,BXM",
        "802,Zach,Zed,82 Fair St,New York,NY,USA,
10033,5185552234,Zap"
    );
    $sth = $dbh->prepare($ins);
    my $row = undef;
    foreach $row ( @rows ) {

```



```
        my @values = split(/,/ , $row);  
        $sth->execute(@values);  
    }  
}
```

END

Python Support

The SAP Sybase IQ Python database interface, `sqlanydb`, is a data access API for the Python language. This section describes how to use SAP Sybase IQ with Python.

sqlanydb

The SQL Anywhere Python database interface (`sqlanydb`) is a data access API for the Python language. The Python Database API specification defines a set of methods that provides a consistent database interface independent of the actual database being used. Using the `sqlanydb` module, your Python scripts have direct access to SAP Sybase IQ database servers.

The `sqlanydb` module implements, with extensions, the Python Database API specification v2.0 written by Marc-André Lemburg. Once you have installed the `sqlanydb` module, you can access and change the information in SAP Sybase IQ databases from Python.

For information about the Python Database API specification v2.0, see <http://www.python.org/dev/peps/pep-0249/>.

The `sqlanydb` module is thread-safe when using Python with threads.

Requirements

The `sqlanydb` module requires the following components.

- Python is required. For a list of supported versions, see <http://www.sybase.com/detail?id=1068981>.
- The `ctypes` module is required. To test if the `ctypes` module is present, open a command prompt window and run Python.

At the Python prompt, enter the following statement.

```
import ctypes
```

If you see an error message, then `ctypes` is not present. The following is an example.

```
>>> import ctypes
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named ctypes
```

If `ctypes` is not included in your Python installation, install it. Installs can be found in the SourceForge.net files section at http://sourceforge.net/project/showfiles.php?group_id=71702.

Peak EasyInstall also installs `ctypes`. To download Peak EasyInstall, go to <http://peak.telecommunity.com/DevCenter/EasyInstall>.

Installing Python Support on Windows

Python support can be set up on Windows by running the applicable setup python script from the SDK\Python subdirectory of your SAP Sybase IQ installation.

Prerequisites

Ensure that Python and the ctypes module are installed. For a list of supported versions of Python, see <http://www.sybase.com/detail?id=1068981>.

Task

1. At a system command prompt, change to the SDK\Python subdirectory of your SAP Sybase IQ installation.
2. Run the following command to install sqlanydb.
3. To test sqlanydb, make a copy of the sample database file in your current directory and run a test.

```
python setup.py install
```

```
newdemo  
cd "%ALLSERPROFILE%\SybaseIQ\demo  
start_iq @iqdemo.cfg iqdemo.db  
python Scripts\test.py
```

The test script makes a connection to the database server and executes a SQL query. If successful, the test displays the message `sqlanydb successfully installed`.

If the tests do not run, ensure that the `bin32` or `bin64` subdirectory of the SAP Sybase IQ installation is in your path.

The `sqlanydb` module is now ready to use.

Installing Python Support on Unix

Python support can be set up on Unix by running the applicable setup python script from the `sdk/python` subdirectory of your SAP Sybase IQ installation.

Prerequisites

Ensure that Python and the ctypes module are installed. For a list of supported versions of Python, see <http://www.sybase.com/detail?id=1068981>.

Task

1. Make sure the environment is set up for SAP Sybase IQ.

Depending on which shell you are using, enter the appropriate command to source the SAP Sybase IQ configuration script from the SAP Sybase IQ installation directory (`bin64` may be used in place of `bin32`, if you have the 64-bit software installed):

In this shell...	Use this command...
sh, ksh, or bash	<code>.bin32/sa_config.sh</code>
csh or tcsh	<code>source bin32/sa_config.csh</code>

2. At a shell prompt, change to the `sdk/python` subdirectory of your SAP Sybase IQ installation.
3. Enter the following command to install `sqlanydb`.
4. To test `sqlanydb`, make a copy of the sample database file in your current directory and run a test.

```
python setup.py install
```

```
newdemo
cd "%ALLSERPROFILE%\SybaseIQ\demo
start_iq @iqdemo.cfg iqdemo.db
python scripts/test.py
```

The test script makes a connection to the database server and executes a SQL query. If successful, the test displays the message `sqlanydb successfully installed`.

If the test does not run, ensure that the `bin32` or `bin64` subdirectory of the SAP Sybase IQ installation is in your path.

The `sqlanydb` module is now ready to use.

Python Scripts That Use `sqlanydb`

This section provides an overview of how to write Python scripts that use the `sqlanydb` interface.

Complete documentation for the API is available online at <http://www.python.org/dev/peps/pep-0249/>.

The `sqlanydb` Module

To use the `sqlanydb` module from a Python script, you must first load it by including the following line at the top of the file.

```
import sqlanydb
```

How to Open and Close a Database Connection Using Python

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the connect method. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The parameters to the connect method are specified as a series of keyword=value pairs delimited by commas.

```
sqlanydb.connect( keyword=value, ...)
```

Some common connection parameters are as follows:

- **DataSourceName="dsn"** – A short form for this connection parameter is `DSN="dsn"`. An example is `DataSourceName="Sybase IQ demo"`.
- **UserID="user-id"** – A short form for this connection parameter is `UID="user-id"`.
- **Password="passwd"** – A short form for this connection parameter is `PWD="passwd"`.
- **DatabaseFile="db-file"** – A short form for this connection parameter is `DBF="db-file"`. An example is `DatabaseFile="iqdemo.db"`.

The following code sample opens and closes a connection to the SAP Sybase IQ sample database. You must start the database server and sample database before running this script.

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( userid="<user_id>",
                       password="<password>" )

# Close the connection
con.close()
```

To avoid starting the database server manually, you could use a data source that is configured to start the server. This is shown in the following example.

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( DSN="Sybase IQ Demo" )

# Close the connection
con.close()
```

How to Obtain Result Sets Using Python

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

The cursor method is used to create a cursor on the open connection. The execute method is used to create a result set. The fetchall method is used to obtain the rows in this result set.

```
import sqlanydb
```

```

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>",
                       password="<password>" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()

```

How to Insert Rows Using Python

The simplest way to insert rows into a table is to use a non-parameterized INSERT statement, meaning that values are specified as part of the SQL statement. A new statement is constructed and executed for each new row. As in the previous example, a cursor is required to execute SQL statements.

The following sample program inserts two new customers into the sample database. Before disconnecting, it commits the transactions to the database.

```

import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>", pwd="<password>" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801, 'Alex', 'Alt', '5 Blue Ave', 'New York', 'NY',
         'USA', '10012', '5185553434', 'BXM'),
        (802, 'Zach', 'Zed', '82 Fair St', 'New York', 'NY',
         'USA', '10033', '5185552234', 'Zap'))

# Set up a SQL INSERT
parms = ("%s", " * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql % rows[0]
cursor.execute(sql % rows[0])
print sql % rows[1]
cursor.execute(sql % rows[1])
cursor.close()

```

```
con.commit()
con.close()
```

An alternate technique is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The executemany method is used to execute an INSERT statement for each member of the set of rows. The new row values are supplied as a single argument to the executemany method.

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>", pwd="<password>" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801, 'Alex', 'Alt', '5 Blue Ave', 'New York', 'NY',
        'USA', '10012', '5185553434', 'BXM'),
        (802, 'Zach', 'Zed', '82 Fair St', 'New York', 'NY',
        'USA', '10033', '5185552234', 'Zap'))

# Set up a parameterized SQL INSERT
parms = ("?", " * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql
cursor.executemany(sql, rows)
cursor.close()
con.commit()
con.close()
```

Although both examples may appear to be equally suitable techniques for inserting row data into a table, the latter example is superior for a couple of reasons. If the data values are obtained by prompts for input, then the first example is susceptible to injection of rogue data including SQL statements. In the first example, the execute method is called for each row to be inserted into the table. In the second example, the executemany method is called only once to insert all the rows into the table.

Database Type Conversion

To control how database types are mapped into Python objects when results are fetched from the database server, conversion callbacks can be registered.

Callbacks are registered using the module level register_converter method. This method is called with the database type as the first parameter and the conversion function as the second parameter. For example, to request that sqlanydb create Decimal objects for data in any column described as having type DT_DECIMAL, you would use the following example:

```
import sqlanydb
import decimal

def convert_to_decimal(num):
    return decimal.Decimal(num)

sqlanydb.register_converter(sqlanydb.DT_DECIMAL,
                             convert_to_decimal)
```


Converters may be registered for the following database types:

```
DT_DATE
DT_TIME
DT_TIMESTAMP
DT_VARCHAR
DT_FIXCHAR
DT_LONGVARCHAR
DT_DOUBLE
DT_FLOAT
DT_DECIMAL
DT_INT
DT_SMALLINT
DT_BINARY
DT_LONGBINARY
DT_TINYINT
DT_BIGINT
DT_UNSMALLINT
DT_UNSBIGINT
DT_BIT
```

The following example demonstrates how to convert decimal results to integer resulting in the truncation of any digits after the decimal point. The salary amount displayed when the application is run is an integral value.

```
import sqlanydb

def convert_to_int(num):
    return int(float(num))

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_int)

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>",
                       password="<password>" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees WHERE EmployeeID=105"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```


PHP Support

PHP provides the ability to retrieve information from many popular databases. SAP Sybase IQ includes a module that provides access to SAP Sybase IQ databases from PHP. You can use the PHP language to retrieve information from SAP Sybase IQ databases and provide dynamic web content on your own web sites.

SAP Sybase IQ PHP Extension

PHP, which stands for PHP: Hypertext Preprocessor, is an open source scripting language. Although it can be used as a general-purpose scripting language, it was designed to be a convenient language in which to write scripts that could be embedded with HTML documents. Unlike scripts written in JavaScript, which are frequently executed by the client, PHP scripts are processed by the web server, and the resulting HTML output sent to the clients. The syntax of PHP is derived from that of other popular languages, such as Java and Perl.

To make it a convenient language in which to develop dynamic web pages, PHP provides the ability to retrieve information from many popular databases, such as SAP Sybase IQ. Included with SAP Sybase IQ is an extension that provides access to SAP Sybase IQ databases from PHP. You can use the SAP Sybase IQ PHP extension and the PHP language to write standalone scripts and create dynamic web pages that rely on information stored in SAP Sybase IQ databases.

The SAP Sybase IQ PHP extension provides a native means of accessing your databases from PHP. You might prefer it to other PHP data access techniques because it is simple, and it helps to avoid system resource leaks that can occur with other techniques.

Prebuilt versions of the PHP extension are provided for Windows, Linux, and Solaris and are installed in the binaries subdirectories of your SAP Sybase IQ installation. Source code for the SAP Sybase IQ PHP extension is installed in the `sdk\php` subdirectory of your SAP Sybase IQ installation.

For more information and the latest SAP Sybase IQ PHP drivers, see <http://www.sybase.com/detail?id=1019698>.

Testing the PHP Extension

Perform a quick check to verify that the SAP Sybase IQ PHP extension is working correctly.

Prerequisites

All of the required PHP components should be installed on your system

Task

1. Make sure that the `bin32` subdirectory of your SAP Sybase IQ installation is in your path. The SAP Sybase IQ PHP extension requires the `bin32` directory to be in your path.
2. At a command prompt, run the following command to start the SAP Sybase IQ sample database.

```
cd "%ALLUSERSPROFILE%" \SybaseIQ\demo
start_iq @iqdemo.cfg iqdemo.db
```

The command starts a database server using the sample database.

3. At a command prompt, change to the `SDK\PHP\Examples` subdirectory of your SAP Sybase IQ installation. Make sure that the `php` executable directory is included in your path. Enter the following command:

```
php test.php
```

Messages similar to the following should appear. If the PHP command is not recognized, verify that PHP is in your path.

```
Installation successful
Using php-5.2.11_sqlanywhere.dll
Connected successfully
```

If the SAP Sybase IQ PHP extension does not load, you can use the command `"php -i"` for helpful information about your PHP setup. Search for `extension_dir` and `sqlanywhere` in the output from this command.

4. When you are done, stop the SAP Sybase IQ database server by clicking **Shut Down** in the database server messages window.

The tests should succeed, indicating that the SAP Sybase IQ PHP extension is working correctly.

Creating and Running PHP Test Pages

Create and run several web pages that test whether PHP is set up properly.

Prerequisites

You must install PHP. For information about installing PHP, see <http://us2.php.net/install>.

Task

This procedure applies to all configurations.

1. Create a file in your root web content directory named `info.php`.

If you are not sure which directory to use, check your web server's configuration file. In Apache installations, the content directory is often called `htdocs`.

2. Insert the following code into this file:

```
<?php phpinfo(); ?>
```

The PHP function, `phpinfo`, generates a page of system setup information. This confirms that your installation of PHP and your web server are working together properly.

3. Copy the file `connect.php` from the `sdk\php\examples` directory to your root web content directory. This confirms that your installation of PHP and SQL Anywhere are working together properly.
4. Create a file in your root web content directory named `sa_test.php` and insert the following code into this file:

```
<?php
    $conn = sasql_connect( "UID=DBA;PWD=sql" );
    $result = sasql_query( $conn, "SELECT * FROM Employees" );
    sasql_result_all( $result );
    sasql_free_result( $result );
    sasql_disconnect( $conn );
?>
```

The `sa_test` page displays the contents of the `Employees` table.

5. Start your web server if it is required.

For example, to start the Apache web server, run the following command from the `bin` subdirectory of your Apache installation:

```
apachectl start
```

6. On Linux, set the SAP Sybase IQ environment variables using one of the supplied scripts.

Depending on which shell you are using, enter the appropriate command to source the SAP Sybase IQ configuration script from your SAP Sybase IQ installation directory:

In this shell...	...use this command
sh, ksh, or bash	<code>. /bin32/sa_config.sh</code>
csh or tcsh	<code>source /bin32/sa_config.csh</code>

7. At a command prompt, start the `iqdemo.db` sample database.
8. To test that PHP and your web server are working correctly with SAP Sybase IQ, access the test pages from a browser that is running on the same computer as the server:

For this test page...	Use this URL...
<code>info.php</code>	<code>http://localhost/info.php</code>
<code>connect.php</code>	<code>http://localhost/connect.php</code>
<code>sa_test.php</code>	<code>http://localhost/sa_test.php</code>

The info page displays the output from the `phpinfo()` call.

The connect page displays the message `Connected successfully`.

The sa_test page displays the contents of the Employees table.

PHP Script Development

This section describes how to write PHP scripts that use the SAP Sybase IQ PHP extension to access SAP Sybase IQ databases.

The source code for these and other examples is located in the `SDK\PHP\Examples` subdirectory of your SAP Sybase IQ installation.

How to Connect to a Database Using PHP

To make a connection to a database, pass a standard SAP Sybase IQ connection string to the database server as a parameter to the `sasql_connect` function. The `<?php` and `?>` tags tell the web server that it should let PHP execute the code that lies between them and replace it with the PHP output.

The source code for this example is contained in your SAP Sybase IQ installation in a file called `connect.php`.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ){
    echo "Connection failed\n";
} else {
    echo "Connected successfully\n";
    sasql_close( $conn );
}?>
```

This script attempts to make a connection to a database on a local server. For this code to succeed, the SAP Sybase IQ sample database or one with identical credentials must be started on a local server.

How to Retrieve Data from a Database Using PHP

One use of PHP scripts in web pages is to retrieve and display information contained in a database. The following examples demonstrate some useful techniques.

Simple select query

The following PHP code demonstrates a convenient way to include the result set of a `SELECT` statement in a web page. This sample is designed to connect to the SAP Sybase IQ sample database and return a list of customers.

This code can be embedded in a web page, provided your web server is configured to execute PHP scripts.

The source code for this sample is contained in your SAP Sybase IQ installation in a file called `query.php`.

```

<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    echo "sasql_connect failed\n";
} else {
    echo "Connected successfully\n";
    # Execute a SELECT statement
    $result = sasql_query( $conn, "SELECT * FROM Customers" );
    if( ! $result ) {
        echo "sasql_query failed!";
    } else {
        echo "query completed successfully\n";
        # Generate HTML from the result set
        sasql_result_all( $result );
        sasql_free_result( $result );
    }
    sasql_close( $conn );
}
?>

```

The `sasql_result_all` function fetches all the rows of the result set and generates an HTML output table to display them. The `sasql_free_result` function releases the resources used to store the result set.

Fetching by column name

In certain cases, you may not want to display all the data from a result set, or you may want to display the data in a different manner. The following sample illustrates how you can exercise greater control over the output format of the result set. PHP allows you to display as much information as you want in whatever manner you choose.

The source code for this sample is contained in your SAP Sybase IQ installation in a file called `fetch.php`.

```

<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    die ("Connection failed");
} else {
    # Connected successfully.
}
# Execute a SELECT statement
$result = sasql_query( $conn, "SELECT * FROM Customers" );
if( ! $result ) {
    echo "sasql_query failed!";
    return 0;
} else {
    echo "query completed successfully\n";
}
# Retrieve meta information about the results
$num_cols = sasql_num_fields( $result );
$num_rows = sasql_num_rows( $result );
echo "Num of rows = $num_rows\n";

```

```

echo "Num of cols = $num_cols\n";
while( ($field = sasql_fetch_field( $result )) ) {
    echo "Field # : $field->id \n";
    echo "\tname    : $field->name \n";
    echo "\tlength  : $field->length \n";
    echo "\ttype    : $field->type \n";
}
# Fetch all the rows
$curr_row = 0;
while( ($row = sasql_fetch_row( $result )) ) {
    $curr_row++;
    $curr_col = 0;
    while( $curr_col < $num_cols ) {
        echo "$row[$curr_col]\t|";
        $curr_col++;
    }
    echo "\n";
}
# Clean up.
sasql_free_result( $result );
sasql_disconnect( $conn );
?>

```

The `sasql_fetch_array` function returns a single row from the table. The data can be retrieved by column names and column indexes.

The `sasql_fetch_assoc` function returns a single row from the table as an associative array. The data can be retrieved by using the column names as indexes. The following is an example.

```

<?php
# Connect using the default user ID and password
$conn = sasql_connect("UID=DBA;PWD=sql");

/* check connection */
if( sasql_errorcode() ) {
    printf("Connect failed: %s\n", sasql_error());
    exit();
}

$query = "SELECT Surname, Phone FROM Employees ORDER by
EmployeeID";

if( $result = sasql_query($conn, $query) ) {

    /* fetch associative array */
    while( $row = sasql_fetch_assoc($result) ) {
        printf ("%s (%s)\n", $row["Surname"], $row["Phone"]);
    }

    /* free result set */
    sasql_free_result($result);
}

/* close connection */
sasql_close($conn);
?>

```


Two other similar methods are provided in the PHP interface: `sasql_fetch_row` returns a row that can be searched by column indexes only, while `sasql_fetch_object` returns a row that can be searched by column names only.

For an example of the `sasql_fetch_object` function, see the `fetch_object.php` example script.

Nested result sets

When a `SELECT` statement is sent to the database, a result set is returned. The `sasql_fetch_row` and `sasql_fetch_array` functions retrieve data from the individual rows of a result set, returning each row as an array of columns that can be queried further.

The source code for this sample is contained in your SAP Sybase IQ installation in a file called `nested.php`.

```
<?php
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( $conn ) {
    // get the GROUPO user id
    $result = sasql_query( $conn,
        "SELECT user_id FROM SYS.SYSUSER " .
        "WHERE user_name='GROUPO'" );
    if( $result ) {
        $row = sasql_fetch_array( $result );
        $user = $row[0];
    } else {
        $user = 0;
    }
    // get the tables created by user GROUPO
    $result = sasql_query( $conn,
        "SELECT table_id, table_name FROM SYS.SYSTABLE " .
        "WHERE creator = $user" );
    if( $result ) {
        $num_rows = sasql_num_rows( $result );
        echo "Returned rows : $num_rows\n";
        while( $row = sasql_fetch_array( $result ) ) {
            echo "Table: $row[1]\n";
            $query = "SELECT table_id, column_name FROM SYS.SYSCOLUMN
" .
                "WHERE table_id = '$row[table_id]'";
            $result2 = sasql_query( $conn, $query );
            if( $result2 ) {
                echo "Columns:";
                while( $detailed = sasql_fetch_array( $result2 ) ) {
                    echo " $detailed[column_name]";
                }
                sasql_free_result( $result2 );
            }
            echo "\n\n";
        }
        sasql_free_result( $result );
    }
    sasql_disconnect( $conn );
}
```

```
}
?>
```

In the above sample, the SQL statement selects the table ID and name for each table from SYSTAB. The `sasql_query` function returns an array of rows. The script iterates through the rows using the `sasql_fetch_array` function to retrieve the rows from an array. An inner iteration goes through the columns of each row and prints their values.

Web Forms

PHP can take user input from a web form, pass it to the database server as a SQL query, and display the result that is returned. The following example demonstrates a simple web form that gives the user the ability to query the sample database using SQL statements and display the results in an HTML table.

The source code for this sample is contained in your SAP Sybase IQ installation in a file called `webisql.php`.

```
<?php
echo "<HTML>\n";
$qname = $_POST["qname"];
$qname = str_replace( "\\\"", "", $qname );
echo "<form method=post action=webisql.php>\n";
echo "<br>Query: <input type=text Size=80 name=qname value=\"\$qname
\">\n";
echo "<input type=submit>\n";
echo "</form>\n";
echo "<HR><br>\n";
if( ! $qname ) {
    echo "No Current Query\n";
    return;
}
# Connect to the database
$con_str =
"UID=<user_id>;PWD=<password>;SERVER=iqdemo;LINKS=tcPIP";
$conn = sasql_connect( $con_str );
if( ! $conn ) {
    echo "sasql_connect failed\n";
    echo "</html>\n";
    return 0;
}
$qname = str_replace( "\\\"", "", $qname );
$result = sasql_query( $conn, $qname );
if( ! $result ) {
    echo "sasql_query failed!";
} else {
    // echo "query completed successfully\n";
    sasql_result_all( $result, "border=1" );
    sasql_free_result( $result );
}
sasql_disconnect( $conn );
echo "</html>\n";
?>
```

This design could be extended to handle complex web forms by formulating customized SQL queries based on the values entered by the user.

BLOBs in PHP Applications

SAP Sybase IQ databases can store any type of data as a binary large object (BLOB). If that data is of a type readable by a web browser, a PHP script can easily retrieve it from the database and display it on a dynamically generated page.

BLOB fields are often used for storing non-text data, such as images in GIF or JPG format. Numerous types of data can be passed to a web browser without any need for third-party software or data type conversion. The following sample illustrates the process of adding an image to the database and then retrieving it again to be displayed in a web browser.

This sample is similar to the sample code in the files `image_insert.php` and `image_retrieve.php` of your SAP Sybase IQ installation. These samples also illustrate the use of a BLOB column for storing images.

```
<?php
    $conn = sasql_connect( "UID=DBA;PWD=sql" )
        or die("Cannot connect to database");
    $create_table = "CREATE TABLE images (ID INTEGER PRIMARY KEY, img
IMAGE)";
    sasql_query( $conn, $create_table);
    $insert = "INSERT INTO images VALUES (99,
xp_read_file('i anywhere_logo.gif'))";
    sasql_query( $conn, $insert );
    $query = "SELECT img FROM images WHERE ID = 99";
    $result = sasql_query($conn, $query);
    $data = sasql_fetch_row($result);
    $img = $data[0];
    header("Content-type: image/gif");
    echo $img;
    sasql_disconnect($conn);
?>
```

To be able to send the binary data from the database directly to a web browser, the script must set the data's MIME type using the header function. In this case, the browser is told to expect a GIF image so it can display it correctly.

How to Build the SAP Sybase IQ PHP Extension on Unix

To connect PHP to SAP Sybase IQ using the SAP Sybase IQ PHP extension on Unix, you must add the SAP Sybase IQ PHP extension's files to PHP's source tree, and then re-compile PHP.

Adding the SAP Sybase IQ PHP Extension Files to the PHP Source Tree on Unix

This topic describes the steps required to add the SAP Sybase IQ PHP extension files to the PHP source tree.

Prerequisites

The following is a list of software you need to have on your system to complete to use the SAP Sybase IQ PHP extension on Unix:

- an SAP Sybase IQ installation, which can run on the same computer as the Apache web server, or on a different computer.
- The source code for the SQL Anywhere PHP extension, which can be downloaded from http://download.sybase.com/iAnywhere/php/2.0.3/src/sasqL_php.zip.
You also need `sqlpp` and `libdblib16.so` (Unix) installed (check your SAP Sybase IQ `lib32` directory).
- The PHP source code, which can be downloaded from <http://www.php.net>.
For a list of supported versions, see <http://www.sybase.com/detail?id=1068981>.
- The Apache web server source code, which can be downloaded from <http://httpd.apache.org>.
If you are going to use a pre-built version of Apache, make sure that you have `apache` and `apache-devel` installed.
- If you plan to use the Unified ODBC PHP extension, you need to have `libdbodbc16.so` (Unix) installed (check your SAP Sybase IQ `lib32` directory).

The following binaries should be installed from your Unix installation disk if they are not already installed, and can be found as RPMs:

```
make
automake
autoconf
makeinfo
bison
gcc
cpp
glibc-devel
kernel-headers
flex
```

You must have the same access privileges as the person who installed PHP to perform certain steps of the installation. Most Unix-based systems offer a `sudo` command that allows users with insufficient permissions to execute certain commands as a user with the right to execute them.

Task

You must have the same access privileges as the person who installed PHP to perform certain steps of the installation. Most Unix-based systems offer a `sudo` command that allows users with insufficient permissions to execute certain commands as a user with the right to execute them.

1. Download the SAP Sybase IQ PHP extension source code from <http://www.sybase.com/detail?id=1019698>. Look for the section entitled **Building the Driver from Source**.
2. From the directory where you saved the SAP Sybase IQ PHP extension, extract the files to the `ext` subdirectory of the PHP source tree:

```
$ tar -xzf sasql_php.zip -C PHP-source-directory/ext/
```

The following example is for PHP version 5.2.11. You must change `php-5.2.11` below to the version of PHP you are using.

```
$ tar -xzf sqlanywhere_php-1.0.8.tar.gz -C ~/php-5.2.11/ext
```

3. Make PHP aware of the extension:

```
$ cd PHP-source-directory/ext/sqlanywhere
$ touch *
$ cd ~/PHP-source-directory
$ ./buildconf
```

The following example is for PHP version 5.2.11. You must change `php-5.2.11` below to the version of PHP you are using.

```
$ cd ~/php-5.2.11/ext/sqlanywhere
$ touch *
$ cd ~/php-5.2.11
$ ./buildconf
```

4. Verify that PHP is aware of the extension:

```
$ ./configure -help | egrep sqlanywhere
```

If you were successful in making PHP aware of the SAP Sybase IQ extension, you should see the following text:

```
--with-sqlanywhere=[DIR]
```

If you are unsuccessful, keep track of the output of this command and post it to the SQL Anywhere Forum at <http://sqlanywhere-forum.sybase.com/> for assistance.

How to Compile Apache and PHP

PHP can be compiled as a shared module of a web server (such as Apache) or as a CGI executable. If you are using a web server that is not supported by PHP, or to execute PHP scripts in a command shell rather than on a web page, you should compile PHP as a CGI executable. Otherwise, to install PHP to operate with Apache, compile it as an Apache module.

Compiling PHP As an Apache Module

To install PHP to operate with Apache, compile it as an Apache module.

Prerequisites

Configure Apache so that it recognizes shared modules using the following steps.

- Configure Apache to recognize shared modules.

Execute commands similar to the following from the directory where your Apache files were extracted:

```
$ cd Apache-source-directory
$ ./configure --enabled-shared=max --enable-module=most --
prefix=/Apache-installation-directory
```

The following example is for Apache version 2.2.9. You must change `apache_2.2.9` to the version of Apache you are using.

```
$ cd ~/apache_2.2.9
$ ./configure --enabled-shared=max --enable-module=most --
prefix=/usr/local/web/apache
```

- Recompile and install the relevant components:

```
$ make
$ make install
```

Now you are ready to compile PHP to operate as an Apache module.

Task

1. Make sure the environment is set up for SAP Sybase IQ.

Depending on which shell you are using, enter the appropriate command from the directory where SAP Sybase IQ is installed.

If you are using this shell...	...use this command
sh, ksh, bash	<code>./IQ_16.sh</code>
csh, tcsh	<code>./IQ_16.csh</code>

2. Configure PHP as an Apache module to include the SAP Sybase IQ PHP extension.

Run the following commands:

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere --with-apxs=/Apache-
installation-directory/bin/apxs
```

The following example is for PHP version 5.2.11. You must change `php-5.2.11` to the version of PHP you are using.

```
$ cd ~/php-5.2.11
```

```
$ ./configure --with-sqlanywhere --with-apxs=/usr/local/web/apache/bin/apxs
```

The `configure` script will try to determine the version and location of your SAP Sybase IQ installation.

3. Recompile the relevant components:

```
$ make
```

4. Check that the libraries are properly linked.

- Linux users (the following example assumes you are using PHP version 5):

```
ldd ../libs/libphp5.so
```

5. Install the PHP binaries in Apache's `lib` directory:

```
$ make install
```

6. Perform verification. PHP does this automatically. All you need is to make sure that your `httpd.conf` configuration file is verified so that Apache will recognize `.php` files as PHP scripts.

`httpd.conf` is stored in the `conf` subdirectory of the Apache directory:

```
$ cd Apache-installation-directory/conf
```

For example:

```
$ cd /usr/local/web/apache/conf
```

Make a backup copy of `httpd.conf` before editing the file (you can replace `pico` with the text editor of your choice):

```
$ cp httpd.conf httpd.conf.backup
$ pico httpd.conf
```

Add or uncomment the following lines in `httpd.conf` (they are not located together in the file):

```
LoadModule php5_module    libexec/libphp5.so
AddModule mod_php5.c
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

The first two lines point Apache to the files that are used for interpreting PHP code, while the other two lines declare file types for files whose extension is `.php` or `.phps` so Apache can recognize and deal with them appropriately.

PHP is successfully compiled as a shared module.

Compiling PHP as a CGI Executable

If you are using a web server that is not supported by PHP, or to execute PHP scripts in a command shell rather than on a web page, you should compile PHP as a CGI executable.

Prerequisites

There are no prerequisites for this task.

Task

1. Make sure the environment is set up for SAP Sybase IQ.

Depending on which shell you are using, enter the appropriate command from the directory where SAP Sybase IQ is installed.

If you are using this shell...	...use this command
sh, ksh, bash	<code>../IQ_16.sh</code>
csh, tcsh	<code>./IQ_16.csh</code>

2. Configure PHP as a CGI executable and with the SAP Sybase IQ PHP extension.

Run the following command from the directory where your PHP files were extracted:

```
$ cd PHP-source-directory  
$ ./configure --with-sqlanywhere
```

The following example is for PHP version 5.2.11. You must change php-5.2.11 to the version of PHP you are using.

```
$ cd ~/php-5.2.11  
$ ./configure --with-sqlanywhere
```

The configuration script will try to determine the version and location of your SAP Sybase IQ installation.

3. Compile the executable:

```
$ make
```

4. Install the components.

```
$ make install
```

PHP is successfully compiled as a CGI executable.

SAP Sybase IQ PHP API Reference

The PHP API supports the following functions:

sasql_affected_rows

Returns the number of rows affected by the last SQL statement. This function is typically used for INSERT, UPDATE, or DELETE statements. For SELECT statements, use the sasql_num_rows function.

Prototype

```
int sasql_affected_rows( sasql_conn $conn )
```


Parameters

\$conn – The connection resource returned by a connect function.

Returns

The number of rows affected.

sasql_commit

Ends a transaction on the SQL Anywhere database and makes any changes made during the transaction permanent. Useful only when the auto_commit option is Off

Prototype

```
bool sasql_commit( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on failure.

sasql_close

Closes a previously opened database connection.

Prototype

```
bool sasql_close( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on failure.

sasql_connect

Establishes a connection to an SAP Sybase IQ database.

Prototype

```
sasql_conn sasql_connect( string $con_str )
```

Parameters

\$con_str – A connection string as recognized by SAP Sybase IQ.

Returns

A positive SAP Sybase IQ connection resource on success, or an error and 0 on failure.

sasql_data_seek

Positions the cursor on row *row_num* on the *\$result* that was opened using *sasql_query*.

Prototype

```
bool sasql_data_seek( sasql_result $result, int row_num )
```

Parameters

\$result – The result resource returned by the *sasql_query* function.

row_num – An integer that represents the new position of the cursor within the result resource. For example, specify 0 to move the cursor to the first row of the result set or 5 to move it to the sixth row. Negative numbers represent rows relative to the end of the result set. For example, -1 moves the cursor to the last row in the result set and -2 moves it to the second-last row.

Returns

TRUE on success or FALSE on error.

sasql_disconnect

Closes a connection that has been opened with *sasql_connect* or *sasql_pconnect*.

Prototype

```
bool sasql_disconnect( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on error.

sasql_error

Returns the error text of the most recently executed SAP Sybase IQ PHP function. Error messages are stored per connection. If no *\$conn* is specified, then *sasql_error* returns the last error message where no connection was available. For example, if you call *sasql_connect* and the connection fails, then call *sasql_error* with no parameter for *\$conn* to get the error message. To obtain the corresponding error code value, use the *sasql_errorcode* function.

Prototype

```
string sasql_error( [ sasql_conn $conn ] )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

A string describing the error.

sasql_errorcode

Returns the error code of the most-recently executed SAP Sybase IQ PHP function. Error codes are stored per connection. If no *\$conn* is specified, then `sasql_errorcode` returns the last error code where no connection was available. For example, if you are calling `sasql_connect` and the connection fails, then call `sasql_errorcode` with no parameter for the *\$conn* to get the error code. To get the corresponding error message use the `sasql_error` function

Prototype

```
int sasql_errorcode( [ sasql_conn $conn ] )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

An integer representing an error code. An error code of 0 means success. A positive error code indicates success with warnings. A negative error code indicates failure.

sasql_escape_string

Escapes all special characters in the supplied string. The special characters that are escaped are `\r`, `\n`, `'`, `"`, `;`, `\`, and the NULL character. This function is an alias of `sasql_real_escape_string`.

Prototype

```
string sasql_escape_string( sasql_conn $conn, string $str )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$string – The string to be escaped.

Returns

The escaped string.

sasql_fetch_array

Fetches one row from the result set. This row is returned as an array that can be indexed by the column names or by the column indexes.

Prototype

```
array sasql_fetch_array( sasql_result $result [, int  
$result_type ])
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

\$result_type – This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants `SASQL_ASSOC`, `SASQL_NUM`, or `SASQL_BOTH`. It defaults to `SASQL_BOTH`.

By using the `SASQL_ASSOC` constant this function will behave identically to the `sasql_fetch_assoc` function, while `SASQL_NUM` will behave identically to the `sasql_fetch_row` function. The final option `SASQL_BOTH` will create a single array with the attributes of both.

Returns

An array that represents a row from the result set, or `FALSE` when no rows are available.

sasql_fetch_assoc

Fetches one row from the result set as an associative array.

Prototype

```
array sasql_fetch_assoc( sasql_result $result )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

Returns

An associative array of strings representing the fetched row in the result set, where each key in the array represents the name of one of the result set's columns or `FALSE` if there are no more rows in resultset.

sasql_fetch_field

Returns an object that contains information about a specific column.

Prototype

```
object sasql_fetch_field( sasql_result $result [, int  
$field_offset ] )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

\$field_offset – An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

Returns

An object that has the following properties:

- **id** – contains the field's number.
- **name** – contains the field's name.
- **numeric** – indicates whether the field is a numeric value.
- **length** – returns the field's native storage size.
- **type** – returns the field's type.
- **native_type** – returns the field's native type. These are values like `DT_FIXCHAR`, `DT_DECIMAL` or `DT_DATE`.
- **precision** – returns the field's numeric precision. This property is only set for fields with `native_type` equal to `DT_DECIMAL`.
- **scale** – returns the field's numeric scale. This property is only set for fields with `native_type` equal to `DT_DECIMAL`.

sasql_fetch_object

Fetches one row from the result set as an object.

Prototype

```
object sasql_fetch_object( sasql_result $result )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

Returns

An object representing the fetched row in the result set where each property name matches one of the result set column names, or `FALSE` if there are no more rows in result set.

sasql_fetch_row

Fetches one row from the result set. This row is returned as an array that can be indexed by the column indexes only.

Prototype

```
array sasql_fetch_row( sasql_result $result )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

Returns

An array that represents a row from the result set, or `FALSE` when no rows are available.

sasql_field_count

Returns the number of columns (fields) the last result contains.

Prototype

```
int sasql_field_count( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

A positive number of columns, or `FALSE` if `$conn` is not valid.

sasql_field_seek

Sets the field cursor to the given offset. The next call to `sasql_fetch_field` will retrieve the field definition of the column associated with that offset.

Prototype

```
bool sasql_field_seek( sasql_result $result, int $field_offset )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

\$field_offset – An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

Returns

TRUE on success or FALSE on error.

sasql_free_result

Frees database resources associated with a result resource returned from `sasql_query`.

Prototype

```
bool sasql_free_result( sasql_result $result )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

Returns

TRUE on success or FALSE on error.

sasql_get_client_info

Returns the version information of the client.

Prototype

```
string sasql_get_client_info( )
```

Parameters

None

Returns

A string that represents the SQL Anywhere client software version. The returned string is of the form X.Y.Z.W where X is the major version number, Y is the minor version number, Z is the patch number, and W is the build number (for example, 10.0.1.3616).

sasql_insert_id

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column. The `sasql_insert_id` function is provided for compatibility with MySQL databases.

Prototype

```
int sasql_insert_id( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

The ID generated for an AUTOINCREMENT column by a previous INSERT statement or zero if last insert did not affect an AUTOINCREMENT column. The function can return FALSE if the *\$conn* is not valid.

sasql_message

Writes a message to the server messages window.

Prototype

```
bool sasql_message( sasql_conn $conn, string $message )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$message – A message to be written to the server messages window.

Returns

TRUE on success or FALSE on failure.

sasql_multi_query

Prepares and executes one or more SQL queries specified by *\$sql_str* using the supplied connection resource. Each query is separated from the other using semicolons. The first query result can be retrieved or stored using `sasql_use_result` or `sasql_store_result`. `sasql_field_count` can be used to check if the query returns a result set or not. All subsequent query results can be processed using `sasql_next_result` and `sasql_use_result/sasql_store_result`.

Prototype

```
bool sasql_multi_query( sasql_conn $conn, string $sql_str )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$sql_str – One or more SQL statements separated by semicolons.

Returns

TRUE on success or FALSE on failure.

sasql_next_result

Prepares the next result set from the last query that executed on *\$conn*.

Prototype

```
bool sasql_next_result( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

FALSE if there is no other result set to be retrieved. TRUE if there is another result to be retrieved. Call `sasql_use_result` or `sasql_store_result` to retrieve the next result set.

sasql_num_fields

Returns the number of fields that a row in the *\$result* contains.

Prototype

```
int sasql_num_fields( sasql_result $result )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

Returns

Returns the number of fields in the specified result set.

sasql_num_rows

Returns the number of rows that the *\$result* contains.

Prototype

```
int sasql_num_rows( sasql_result $result )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

Returns

A positive number if the number of rows is exact, or a negative number if it is an estimate. To get the exact number of rows, the database option `row_counts` must be set permanently on the database, or temporarily on the connection.

sasql_pconnect

Establishes a persistent connection to an SAP Sybase IQ database. Because of the way Apache creates child processes, you may observe a performance gain when using `sasql_pconnect` instead of `sasql_connect`. Persistent connections may provide improved performance in a similar fashion to connection pooling. If your database server has a limited number of connections (for example, the personal database server is limited to 10 concurrent connections), caution should be exercised when using persistent connections. Persistent connections could be attached to each of the child processes, and if you have more child processes in Apache than there are available connections, you will receive connection errors.

Prototype

```
sasql_conn sasql_pconnect( string $con_str )
```

Parameters

\$con_str – A connection string as recognized by SAP Sybase IQ.

Returns

A positive SAP Sybase IQ persistent connection resource on success, or an error and 0 on failure.

sasql_prepare

Prepares the supplied SQL string.

Prototype

```
sasql_stmt sasql_prepare( sasql_conn $conn, string $sql_str )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$sql_str – The SQL statement to be prepared. The string can include parameter markers by embedding question marks at the appropriate positions.

Returns

A statement object or FALSE on failure.

sasql_query

Prepares and executes the SQL query `$sql_str` on the connection identified by `$conn` that has already been opened using `sasql_connect` or `sasql_pconnect`. The `sasql_query` function is

equivalent to calling two functions, `sasql_real_query` and one of `sasql_store_result` or `sasql_use_result`.

Prototype

```
mixed sasql_query( sasql_conn $conn, string $sql_str [, int
$result_mode ] )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$sql_str – A SQL statement supported by SQL Anywhere.

\$result_mode – Either `SASQL_USE_RESULT`, or `SASQL_STORE_RESULT` (the default).

Returns

FALSE on failure; TRUE on success for INSERT, UPDATE, DELETE, CREATE; `sasql_result` for SELECT.

sasql_real_escape_string

Escapes all special characters in the supplied string. The special characters that are escaped are `\r`, `\n`, `'`, `"`, `;`, `\`, and the NULL character.

Prototype

```
string sasql_real_escape_string( sasql_conn $conn, string $str )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$string – The string to be escaped.

Returns

The escaped string or FALSE on error.

sasql_real_query

Executes a query against the database using the supplied connection resource. The query result can be retrieved or stored using `sasql_store_result` or `sasql_use_result`. The `sasql_field_count` function can be used to check if the query returns a result set or not. The `sasql_query` function is equivalent to calling this function and one of `sasql_store_result` or `sasql_use_result`.

Prototype

```
bool sasql_real_query( sasql_conn $conn, string $sql_str )
```

Parameters

\$conn – The connection resource returned by a connect function.

\$sql_str – A SQL statement supported by SQL Anywhere.

Returns

TRUE on success or FALSE on failure.

sasql_result_all

Fetches all results of the *\$result* and generates an HTML output table with an optional formatting string.

Prototype

```
bool sasql_result_all( resource $result
[, $html_table_format_string
[, $html_table_header_format_string
[, $html_table_row_format_string
[, $html_table_cell_format_string
] ] ] ] )
```

Parameters

\$result – The result resource returned by the `sasql_query` function.

\$html_table_format_string – A format string that applies to HTML tables. For example, "border=1; cellpadding=5". The special value none does not create an HTML table. This is useful to customize your column names or scripts. To avoid specifying an explicit value for this parameter, use NULL for the parameter value.

\$html_table_header_format_string – A format string that applies to column headings for HTML tables. For example, "bgcolor=#FF9533". The special value none does not create an HTML table. This is useful to customize your column names or scripts. To avoid specifying an explicit value for this parameter, use NULL for the parameter value.

\$html_table_row_format_string – A format string that applies to rows within HTML tables. For example, "onclick='alert('this')' ". If you would like different formats that alternate, use the special token <>. The left side of the token indicates which format to use on odd rows and the right side of the token is used to format even rows. If you do not place this token in your format string, all rows have the same format. If you do not want to specify an explicit value for this parameter, use NULL for the parameter value.

\$html_table_cell_format_string – A format string that applies to cells within HTML table rows. For example, "onclick='alert('this')' ". If you do not want to specify an explicit value for this parameter, use NULL for the parameter value.

Returns

TRUE on success or FALSE on failure.

sasql_rollback

Ends a transaction on the database and discards any changes made during the transaction. This function is only useful when the `auto_commit` option is Off.

Prototype

```
bool sasql_rollback( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on failure.

sasql_set_option

Sets the value of the specified option on the specified connection.

Prototype

```
bool sasql_set_option( sasql_conn $conn, string $option, mixed $value )
```

Description

You can set the value for the following options:

Name	Description	Default
auto_commit	When this option is set to on, the database server commits after executing each statement.	on
row_counts	When this option is set to FALSE, the <code>sasql_num_rows</code> function returns an estimate of the number of rows affected. To obtain an exact count, set this option to TRUE.	FALSE

Name	Description	Default
verbose_errors	When this option is set to TRUE, the PHP driver returns verbose errors. When this option is set to FALSE, you must call the sasql_error or sasql_errrcode functions to get further error information.	TRUE

You can change the default value for an option by including the following line in the `php.ini` file. In this example, the default value is set for the `auto_commit` option.

```
sqlanywhere.auto_commit=0
```

Parameters

\$conn – The connection resource returned by a connect function.

\$option – The name of the option you want to set.

\$value – The new option value.

Returns

TRUE on success or FALSE on failure.

sasql_stmt_affected_rows

Returns the number of rows affected by executing the statement.

Prototype

```
int sasql_stmt_affected_rows( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource that was executed by `sasql_stmt_execute`.

Returns

The number of rows affected or FALSE on failure.

sasql_stmt_bind_param

Binds PHP variables to statement parameters.

Prototype

```
bool sasql_stmt_bind_param( sasql_stmt $stmt, string $types,
mixed &$var_1 [, mixed &$var_2 .. ] )
```

Parameters

\$stmt – A prepared statement resource that was returned by the `sasql_prepare` function.

\$types – A string that contains one or more characters specifying the types of the corresponding bind. This can be any of: `s` for string, `i` for integer, `d` for double, `b` for blobs. The length of the `$types` string must match the number of parameters that follow the `$types` parameter (`$var_1`, `$var_2`, ...). The number of characters should also match the number of parameter markers (question marks) in the prepared statement.

\$var_n – The variable references.

Returns

TRUE if binding the variables was successful or FALSE otherwise.

sasql_stmt_bind_param_ex

Binds a PHP variable to a statement parameter.

Prototype

```
bool sasql_stmt_bind_param_ex( sasql_stmt $stmt, int
    $param_number, mixed &$var, string $type [, bool $is_null [, int
    $direction ] ] )
```

Parameters

\$stmt – A prepared statement resource that was returned by the `sasql_prepare` function.

\$param_number – The parameter number. This should be a number between 0 and (`sasql_stmt_param_count($stmt) - 1`).

\$var – A PHP variable. Only references to PHP variables are allowed.

\$type – Type of the variable. This can be one of: `s` for string, `i` for integer, `d` for double, `b` for blobs.

\$is_null – Whether the value of the variable is NULL or not.

\$direction – Can be `SASQL_D_INPUT`, `SASQL_D_OUTPUT`, or `SASQL_INPUT_OUTPUT`.

Returns

TRUE if binding the variable was successful or FALSE otherwise.

sasql_stmt_bind_result

Binds one or more PHP variables to result columns of a statement that was executed, and returns a result set.

Prototype

```
bool sasql_stmt_bind_result( sasql_stmt $stmt, mixed &$var1 [,
mixed &$var2 .. ] )
```

Parameters

\$stmt – A statement resource that was executed by `sasql_stmt_execute`.

\$var1 – References to PHP variables that will be bound to result set columns returned by the `sasql_stmt_fetch`.

Returns

TRUE on success or FALSE on failure.

sasql_stmt_close

Closes the supplied statement resource and frees any resources associated with it. This function will also free any result objects that were returned by the `sasql_stmt_result_metadata`.

Prototype

```
bool sasql_stmt_close( sasql_stmt $stmt )
```

Parameters

\$stmt – A prepared statement resource that was returned by the `sasql_prepare` function.

Returns

TRUE for success or FALSE on failure.

sasql_stmt_data_seek

This function seeks to the specified offset in the result set.

Prototype

```
bool sasql_stmt_data_seek( sasql_stmt $stmt, int $offset )
```

Parameters

\$stmt – A statement resource.

\$offset – The offset in the result set. This is a number between 0 and (sasql_stmt_num_rows(\$stmt) - 1).

Returns

TRUE on success or FALSE failure.

sasql_stmt_errno

Returns the error code for the most recently executed statement function using the specified statement resource.

Prototype

```
int sasql_stmt_errno( sasql_stmt $stmt )
```

Parameters

\$stmt – A prepared statement resource that was returned by the sasql_prepare function.

Returns

An integer error code.

sasql_stmt_error

Returns the error text for the most recently executed statement function using the specified statement resource.

Prototype

```
string sasql_stmt_error( sasql_stmt $stmt )
```

Parameters

\$stmt – A prepared statement resource that was returned by the sasql_prepare function.

Returns

A string describing the error.

sasql_stmt_execute

Executes the prepared statement. The sasql_stmt_result_metadata can be used to check whether the statement returns a result set.

Prototype

```
bool sasql_stmt_execute( sasql_stmt $stmt )
```

Parameters

\$stmt – A prepared statement resource that was returned by the `sasql_prepare` function. Variables should be bound before calling `execute`.

Returns

TRUE for success or FALSE on failure.

sasql_stmt_fetch

This function fetches one row out of the result for the statement and places the columns in the variables that were bound using `sasql_stmt_bind_result`.

Prototype

```
bool sasql_stmt_fetch( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource.

Returns

TRUE on success or FALSE on failure.

sasql_stmt_field_count

This function returns the number of columns in the result set of the statement.

Prototype

```
int sasql_stmt_field_count( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource.

Returns

The number of columns in the result of the statement. If the statement does not return a result, it returns 0.

sasql_stmt_free_result

This function frees cached result set of the statement.

Prototype

```
bool sasql_stmt_free_result( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource that was executed using `sasql_stmt_execute`.

Returns

TRUE on success or FALSE on failure.

sasql_stmt_insert_id

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

Prototype

```
int sasql_stmt_insert_id( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource that was executed by `sasql_stmt_execute`.

Returns

The ID generated for an IDENTITY column or a DEFAULT AUTOINCREMENT column by a previous INSERT statement, or zero if the last insert did not affect an IDENTITY or DEFAULT AUTOINCREMENT column. The function can return FALSE (0) if \$stmt is not valid.

sasql_stmt_next_result

This function advances to the next result from the statement. If there is another result set, the currently cached results are discarded and the associated result set object deleted (as returned by `sasql_stmt_result_metadata`).

Prototype

```
bool sasql_stmt_next_result( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource.

Returns

TRUE on success or FALSE failure.

sasql_stmt_num_rows

Returns the number of rows in the result set. The actual number of rows in the result set can only be determined after the `sasql_stmt_store_result` function is called to buffer the entire result set. If the `sasql_stmt_store_result` function has not been called, 0 is returned.

Prototype

```
int sasql_stmt_num_rows( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource that was executed by `sasql_stmt_execute` and for which `sasql_stmt_store_result` was called.

Returns

The number of rows available in the result or 0 on failure.

sasql_stmt_param_count

Returns the number of parameters in the supplied prepared statement resource.

Prototype

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource returned by the `sasql_prepare` function.

Returns

The number of parameters or FALSE on error.

sasql_stmt_reset

This function resets the `$stmt` object to the state just after the describe. Any variables that were bound are unbound and any data sent using `sasql_stmt_send_long_data` are dropped.

Prototype

```
bool sasql_stmt_reset( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource.

Returns

TRUE on success or FALSE on failure.

sasql_stmt_result_metadata

Returns a result set object for the supplied statement.

Prototype

```
sasql_result sasql_stmt_result_metadata( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource that was prepared and executed.

Returns

sasql_result object or FALSE if the statement does not return any results.

sasql_stmt_send_long_data

Allows the user to send parameter data in chunks. The user must first call `sasql_stmt_bind_param` or `sasql_stmt_bind_param_ex` before attempting to send any data. The bind parameter must be of type string or blob. Repeatedly calling this function appends on to what was previously sent.

Prototype

```
bool sasql_stmt_send_long_data( sasql_stmt $stmt, int  
$param_number, string $data )
```

Parameters

\$stmt – A statement resource that was prepared using `sasql_prepare`.

\$param_number – The parameter number. This must be a number between 0 and `(sasql_stmt_param_count($stmt) - 1)`.

\$data – The data to be sent.

Returns

TRUE on success or FALSE on failure.

sasql_stmt_store_result

This function allows the client to cache the whole result set of the statement. You can use the function `sasql_stmt_free_result` to free the cached result.

Prototype

```
bool sasql_stmt_store_result( sasql_stmt $stmt )
```

Parameters

\$stmt – A statement resource that was executed using `sasql_stmt_execute`.

Returns

TRUE on success or FALSE on failure.

sasql_store_result

Transfers the result set from the last query on the database connection `$conn` to be used with the `sasql_data_seek` function.

Prototype

```
boolean sasql_store_result( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

FALSE if the query does not return a result object, or a result set object, that contains all the rows of the result. The result is cached at the client.

sasql_sqlstate

Returns the most recent SQLSTATE string. SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition. SQLSTATE codes consists of five characters with "00000" representing no error. The values are defined by the ISO/ANSI SQL standard.

Prototype

```
string sasql_sqlstate( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

Returns a string of five characters containing the current SQLSTATE code. The result "00000" means no error.

sasql_use_result

Initiates a result set retrieval for the last query that executed on the connection.

Prototype

```
sasql_result sasql_use_result( sasql_conn $conn )
```

Parameters

\$conn – The connection resource returned by a connect function.

Returns

FALSE if the query does not return a result object or a result set object. The result is not cached on the client.

Ruby Support

There are three different Ruby Application Programming Interfaces supported by SAP Sybase IQ. First, there is the SAP Sybase IQ Ruby API. This API provides a Ruby wrapping over the interface exposed by the SAP Sybase IQ C API. Second, there is support for ActiveRecord, an object-relational mapper popularized by being part of the Ruby on Rails web development framework. Third, there is support for Ruby DBI. SAP Sybase IQ provides a Ruby Database Driver (DBD) which can be used with DBI.

Ruby API Support

There are three separate packages available in the SAP Sybase IQ Ruby project. The simplest way to install any of these packages is to use RubyGems.

The home for the SAP Sybase IQ Ruby project is <http://sqlanywhere.rubyforge.org/>.

Native Ruby Driver

sqlanywhere – This package is a low-level driver that allows Ruby code to interface with SAP Sybase IQ databases. This package provides a Ruby wrapping over the interface exposed by the SAP Sybase IQ C API. This package is written in C and is available as source, or as pre-compiled gems, for Windows and Linux. If you have RubyGems installed, this package can be obtained by running the following command:

```
gem install sqlanywhere
```

This package is a prerequisite for any of the other SQL Anywhere Ruby packages.

Rails

Rails is a web development framework written in the Ruby language. Its strength is in web application development. A familiarity with the Ruby programming language is highly recommended before you attempt Rails development. See <http://www.rubyonrails.org/>.

ActiveRecord Adapter

activerecord-sqlanywhere-adapter – This package is an adapter that allows ActiveRecord to communicate with SAP Sybase IQ. ActiveRecord is an object-relational mapper, popularized by being part of the Ruby on Rails web development framework. This package is written in pure Ruby, and available in source, or gem format. This adapter uses (and has a dependency on) the `sqlanywhere` gem. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install activerecord-sqlanywhere-adapter
```

SQL Anywhere Ruby/DBI Driver

dbi – This package is a DBI driver for Ruby. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install dbi
```

dbd-sqlanywhere – This package is a driver that allows Ruby/DBI to communicate with SAP Sybase IQ. Ruby/DBI is a generic database interface modeled after the popular Perl DBI module. This package is written in pure Ruby, and available in source, or gem format. This driver uses (and has a dependency on) the sqlanywhere gem. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install dbd-sqlanywhere
```

For feedback on any of these packages, use the mailing list *sqlanywhere-users@rubyforge.com*.

Configuring Rails Support in SAP Sybase IQ

You can configure Ruby on Rails support in SAP Sybase IQ.

Prerequisites

There are no prerequisites for this task.

Task

1. Install RubyGems. It simplifies the installation of Ruby packages. The Ruby on Rails download page directs you to the correct version to install. See <http://www.rubyonrails.org/>.
2. Install the Ruby interpreter on your system. The Ruby on Rails download page recommends which version to install. See <http://www.rubyonrails.org/>.
3. Install Ruby Rails and its dependencies by running the following command:

```
gem install rails
```

4. Install the Ruby Development Kit (DevKit). Download the DevKit from <http://rubyinstaller.org/downloads/> and follow the instructions at <http://github.com/oneclick/rubyinstaller/wiki/Development-Kit>.
5. Install the SQL Anywhere ActiveRecord support (activerecord-sqlanywhere-adapter) by running the following command:

```
gem install activerecord-sqlanywhere-adapter
```

6. Add SAP Sybase IQ to the set of database management systems supported by Rails. At the time of writing, Rails 3.1.3 was the current released version.
 - a. Configure a database by creating a `sqlanywhere.yml` file in the Rails `configs\databases` directory. If you have installed Ruby in the `\Ruby` directory and you have installed version 3.1.3 of Rails, then the path to this file would be `\Ruby\lib`

`\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\rails\app\templates\config\databases`. The contents of this file should be:

```
#
# SQL Anywhere database configuration
#
# This configuration file defines the patten used for
# database filenames. If your application is called "blog",
# then the database names will be blog_development,
# blog_test, blog_production. The specified username and
# password should permit DBA access to the database.
#

development:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_development
  username: DBA
  password: sql

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run
# "rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_test
  username: DBA
  password: sql

production:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_production
  username: DBA
  password: sql
```

The `sqlanywhere.yml` file provides a template for creating `database.yml` files in Rails projects. The following database options can be specified:

- **adapter** – (required, no default). This option must be set to `sqlanywhere` to use the SQL Anywhere ActiveRecord adapter.
- **database** – (required, no default). This option corresponds to `DatabaseName` in a connection string.
- **server** – (optional, defaults to the `database` option). This option corresponds to `ServerName` in a connection string.
- **username** – (optional, defaults to 'DBA'). This option corresponds to `UserID` in a connection string.
- **password** – (optional, defaults to 'sql'). This option corresponds to `Password` in a connection string.

- **encoding** – (optional, defaults to the OS character set). This option corresponds to CharSet in a connection string.
 - **commlinks** – (optional). This option corresponds to CommLinks in a connection string.
 - **connection_name** – (optional). This option corresponds to ConnectionName in connection string.
- b. Update the Rails `app_base.rb` file. Using the same assumptions in the previous step, this file is located in the path `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\app_base.rb`. Edit the `app_base.rb` file and locate the following line:

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase  
ibm_db sqlserver )
```

Add `sqlanywhere` to the list as follows:

```
DATABASES = %w( sqlanywhere mysql oracle postgresql sqlite3  
frontbase ibm_db sqlserver )
```

7. Follow the tutorial on the Ruby on Rails web site (http://guides.rails.info/getting_started.html) using the following SAP Sybase IQ-specific notes:
- In the tutorial, you are shown the command to initialize the `blog` project. Here is the command to initialize the `blog` project for use with SAP Sybase IQ:

```
rails new blog -d sqlanywhere
```

- After you create the `blog` application, switch to its folder to continue work directly in that application:

```
cd blog
```

- Edit the `gemfile` file to include a `gem` directive for the SQL Anywhere ActiveRecord adapter. Add the new directive following the indicated line below:

```
gem 'sqlanywhere'  
gem 'activerecord-sqlanywhere-adapter'
```

- The `config\database.yml` file references the development, test, and production databases. Instead of using a `rake` command to create the databases as indicated by the tutorial, change to the `db` directory of the project and create three databases as follows.

```
cd db  
iqinit -dba DBA,sql blog_development  
iqinit -dba DBA,sql blog_test  
iqinit -dba DBA,sql blog_production  
cd ..
```

You have configured Rails support in SAP Sybase IQ.

Next

Start the database server and the three databases as follows.

```
iqsrv16 -n blog blog_development.db blog_production.db blog_test.db
```

The database server name in the command line (`blog`) must match the name specified by the `server:tags` in the `database.yml` file. The `sqlanywhere.yml` template file is configured to ensure that the database server name matches the project name in all generated `database.yml` files.

Ruby-DBI Driver

This section provides an overview of how to write Ruby applications that use the DBI driver.

Loading the DBI Module

To use the DBI:SQLAnywhere interface from a Ruby application, you must first tell Ruby that you plan to use the Ruby DBI module. To do so, include the following line near the top of the Ruby source file.

```
require 'dbi'
```

The DBI module automatically loads the SQLAnywhere database driver (DBD) interface as required.

Opening and Closing a Connection

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the `connect` function. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The call to the `connect` function takes the general form:

```
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password,
options)
```

- ***server-name*** – is the name of the database server that you want to connect to. Alternately, you can specify a connection string in the format "option1=value1;option2=value2;...".
- ***user-id*** – is a valid user ID. Unless this string is empty, ";UID=value" is appended to the connection string.
- ***password*** – is the corresponding password for the user ID. Unless this string is empty, ";PWD=value" is appended to the connection string.
- ***options*** – is a hash of additional connection parameters such as `DatabaseName`, `DatabaseFile`, and `ConnectionName`. These are appended to the connection string in the format "option1=value1;option2=value2;...".

To demonstrate the `connect` function, make the `iqdemo` database, start the database server and `iqdemo` database before running the sample Ruby scripts.

```
$IQDIR16/demo
  mkiqdemo.sh
start_iq @iqdemo.cfg iqdemo.db
```

The following code sample opens and closes a connection to the `iqdemo` database. The string "myserver" in the example below is the server name.

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', '<user_id>', '<password>')
```

```
do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

Optionally, you can specify a connection string in place of the server name. For example, in the above script may be altered by replacing the first parameter to the connect function as follows:

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:SERVER=myserver;DBN=iqdemo',
'<user_id>', '<password>') do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

The user ID and password cannot be specified in the connection string. Ruby DBI will automatically fill in the username and password with defaults if these arguments are omitted, so you should never include a UID or PWD connection parameter in your connection string. If you do, an exception will be thrown.

The following example shows how additional connection parameters can be passed to the connect function as a hash of key/value pairs.

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', '<user_id>', '<password>',
  { :ConnectionName => "RubyDemo",
    :DatabaseFile => "iqdemo.db",
    :DatabaseName => "iqdemo" }
  ) do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

Selecting Data

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

A SQL statement must be executed first. If the statement returns a result set, you use the resulting statement handle to retrieve meta information about the result set and the rows of the result set. The following example obtains the column names from the metadata and displays the column names and values for each row fetched.

```
require 'dbi'

def db_query( dbh, sql )
  sth = dbh.execute(sql)
  print "# of Fields: #{sth.column_names.size}\n"
```

```

    sth.fetch do |row|
      print "\n"
      sth.column_info.each_with_index do |info, i|
        unless info["type_name"] == "LONG VARBINARY"
          print "#{info["name"]}#{row[i]}\n"
        end
      end
    end
  end
  sth.finish
end

begin
  dbh = DBI.connect('DBI:SQLAnywhere:demo', '<user_id>',
    '<password>')
  db_query(dbh, "SELECT * FROM Products")
rescue DBI::DatabaseError => e
  puts "An error occurred"
  puts "Error code: #{e.err}"
  puts "Error message: #{e.errstr}"
  puts "Error SQLSTATE: #{e.state}"
ensure
  dbh.disconnect if dbh
end

```

The first few lines of output that appear are reproduced below.

```

# of Fields:  8

ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00

ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00

```

It is important to call `finish` to release the statement handle when you are done. If you do not, then you may get an error like the following:

```
Resource governor for 'prepared statements' exceeded
```

To detect handle leaks, the SAP Sybase IQ database server limits the number of cursors and prepared statements permitted to a maximum of 50 per connection by default. The resource governor automatically generates an error if these limits are exceeded. If you get this error, check for undestroyed statement handles. Use `prepare_cached` sparingly, as the statement handles are not destroyed.

If necessary, you can alter these limits by setting the `max_cursor_count` and `max_statement_count` options.

Inserting Rows

Inserting rows requires a handle to an open connection. The simplest way to insert rows is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The statement is first prepared, and then executed once per new row. The new row values are supplied as parameters to the `execute` method.

```
require 'dbi'

def db_query( dbh, sql )
  sth = dbh.execute(sql)
  print "# of Fields: #{sth.column_names.size}\n"
  sth.fetch do |row|
    print "\n"
    sth.column_info.each_with_index do |info, i|
      unless info["type_name"] == "LONG VARBINARY"
        print "#{info["name"]}#{row[i]}\n"
      end
    end
  end
  sth.finish
end

def db_insert( dbh, rows )
  sql = "INSERT INTO Customers (ID, GivenName, Surname,
    Street, City, State, Country, PostalCode,
    Phone, CompanyName)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
  sth = dbh.prepare(sql);
  rows.each do |row|
    sth.execute(row[0],row[1],row[2],row[3],row[4],
      row[5],row[6],row[7],row[8],row[9])
  end
end

begin
  dbh = DBI.connect('DBI:SQLAnywhere:demo', '<user_id>',
  '<password>')
  rows = [
    [801, 'Alex', 'Alt', '5 Blue Ave', 'New York', 'NY', 'USA',
      '10012', '5185553434', 'BXM'],
    [802, 'Zach', 'Zed', '82 Fair St', 'New York', 'NY', 'USA',
      '10033', '5185552234', 'Zap']
  ]
  db_insert(dbh, rows)
  dbh.commit
  db_query(dbh, "SELECT * FROM Customers WHERE ID > 800")
rescue DBI::DatabaseError => e
  puts "An error occurred"
  puts "Error code: #{e.err}"
  puts "Error message: #{e.errstr}"
  puts "Error SQLSTATE: #{e.state}"
end
```



```
ensure
  dbh.disconnect if dbh
end
```

SAP Sybase IQ Ruby API Reference

SAP Sybase IQ provides a low-level interface to the SAP Sybase IQ C API. The API described in the following sections permits the rapid development of SQL applications. To demonstrate the power of Ruby application development, consider the following sample Ruby program. It loads the SAP Sybase IQ Ruby extension, connects to the sample database, lists column values from the Products table, disconnects, and terminates.

```
begin
  require 'rubygems'
  gem 'sqlanywhere'
  unless defined? SQLAnywhere
    require 'sqlanywhere'
  end
end
api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
conn = api.sqlany_new_connection()
api.sqlany_connect( conn, "DSN=Sybase IQ Demo" )
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Products" )
num_rows = api.sqlany_num_rows( stmt )
num_rows.times {
  api.sqlany_fetch_next( stmt )
  num_cols = api.sqlany_num_cols( stmt )
  for col in 1..num_cols do
    info = api.sqlany_get_column_info( stmt, col - 1 )
    unless info[3]==1 # Don't do binary
      rc, value = api.sqlany_get_column( stmt, col - 1 )
      print "#{info[2]}=#{value}\n"
    end
  end
  print "\n"
}
api.sqlany_free_stmt( stmt )
api.sqlany_disconnect(conn)
api.sqlany_free_connection(conn)
api.sqlany_fini()
SQLAnywhere::API.sqlany_finalize_interface( api )
```

The first two rows of the result set output from this Ruby program are shown below:

```
ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00
```

```
ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

The following sections describe each of the supported functions.

sqlany_affected_rows

Returns the number of rows affected by execution of the prepared statement.

Syntax

```
sqlany_affected_rows ( $stmt )
```

Parameters

- **\$stmt** – A statement that was prepared and executed successfully in which no result set was returned. For example, an INSERT, UPDATE or DELETE statement was executed.

Returns

Returns a scalar value that is the number of rows affected, or -1 on failure.

Example

```
affected = api.sqlany_affected( stmt )
```

sqlany_bind_param Function

Binds a user-supplied buffer as a parameter to the prepared statement.

Syntax

```
sqlany_bind_param ( $stmt, $index, $param )
```

Parameters

- **\$stmt** – A statement object returned by the successful execution of `sqlany_prepare`.
- **\$index** – The index of the parameter. The number must be between 0 and `sqlany_num_params() - 1`.
- **\$param** – A filled bind object retrieved from `sqlany_describe_bind_param`.

Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
```

```
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_clear_error Function

Clears the last stored error code.

Syntax

```
sqlany_clear_error ( $conn )
```

Parameters

- **\$conn** – A connection object returned from `sqlany_new_connection`.

Returns

Returns nil.

Example

```
api.sqlany_clear_error( conn )
```

sqlany_client_version Function

Returns the current client version.

Syntax

```
sqlany_client_version ( )
```

Returns

Returns a scalar value that is the client version string.

Example

```
buffer = api.sqlany_client_version()
```

sqlany_commit Function

Commits the current transaction.

Syntax

```
sqlany_commit ( $conn )
```

Parameters

- **\$conn** – The connection object on which the commit operation is to be performed.

Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

Example

```
rc = api.sqlany_commit( conn )
```

sqlany_connect Function

Creates a connection to a SQL Anywhere database server using the specified connection object and connection string.

Syntax

```
sqlany_connect ( $conn, $str )
```

Parameters

- **\$conn** – The connection object created by `sqlany_new_connection`.
- **\$str** – A SQL Anywhere connection string.

Returns

Returns a scalar value that is 1 if the connection is established successfully or 0 when the connection fails. Use `sqlany_error` to retrieve the error code and message.

Example

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Connection status = #{status}\n"
```

sqlany_describe_bind_param Function

Describes the bind parameters of a prepared statement.

Syntax

```
sqlany_describe_bind_param ( $stmt, $index )
```

Parameters

- **\$stmt** – A statement prepared successfully using `sqlany_prepare`.
- **\$index** – The index of the parameter. The number must be between 0 and `sqlany_num_params()` - 1.

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and a described parameter as the second element.

Remarks

This function allows the caller to determine information about prepared statement parameters. The type of prepared statement (stored procedure or a DML), determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_disconnect Function

Disconnects a SQL Anywhere connection. All uncommitted transactions are rolled back.

Syntax

```
sqlany_disconnect ( $conn )
```

Parameters

- **\$conn** – A connection object with a connection established using `sqlany_connect`.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}\n"
```

sqlany_error Function

Returns the last error code and message stored in the connection object.

Syntax

```
sqlany_error ( $conn )
```

Parameters

- **\$conn** – A connection object returned from `sqlany_new_connection`.

Returns

Returns a 2-element array that contains the SQL error code as the first element and an error message string as the second element.

For the error code, positive values are warnings, negative values are errors, and 0 is success.

Example

```
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

sqlany_execute Function

Executes a prepared statement.

Syntax

```
sqlany_execute ( $stmt )
```

Parameters

- **\$stmt** – A statement prepared successfully using `sqlany_prepare`.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Remarks

You can use `sqlany_num_cols` to verify if the statement returned a result set.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_execute_direct Function

Executes the SQL statement specified by the string argument.

Syntax

```
sqlany_execute_direct ( $conn, $sql )
```

Parameters

- **\$conn** – A connection object with a connection established using `sqlany_connect`.
- **\$sql** – A SQL string. The SQL string should not have parameters such as ?.

Returns

Returns a statement object or nil on failure.

Remarks

Use this function to prepare and execute a statement in one step. Do not use this function to execute a SQL statement with parameters.

Example

```

stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"

```

sqlany_execute_immediate Function

Executes the specified SQL statement immediately without returning a result set. It is useful for statements that do not return result sets.

Syntax

```
sqlany_execute_immediate ( $conn, $sql )
```

Parameters

- **\$conn** – A connection object with a connection established using `sqlany_connect`.
- **\$sql** – A SQL string. The SQL string should not have parameters such as `?`.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```

rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= 50" )

```

sqlany_fetch_absolute Function

Moves the current row in the result set to the row number specified and then fetches the data at that row.

Syntax

```
sqlany_fetch_absolute ( $stmt, $row_num )
```

Parameters

- **\$stmt** – A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.
- **\$row_num** – The row number to be fetched. The first row is 1, the last row is -1.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_absolute( stmt, 2 )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_fetch_next Function

Returns the next row from the result set. This function first advances the row pointer and then fetches the data at the new row.

Syntax

```
sqlany_fetch_next ( $stmt )
```

Parameters

- **\$stmt** – A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```


sqlany_fini Function

Frees resources allocated by the API.

Syntax

```
sqlany_fini ( )
```

Returns

Returns nil.

Example

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_connection Function

Frees the resources associated with a connection object.

Syntax

```
sqlany_free_connection ( $conn )
```

Parameters

- **\$conn** – A connection object created by `sqlany_new_connection`.

Returns

Returns nil.

Example

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_stmt Function

Frees resources associated with a statement object.

Syntax

```
sqlany_free_stmt ( $stmt )
```

Parameters

- **\$stmt** – A statement object returned by the successful execution of `sqlany_prepare` or `sqlany_execute_direct`.

Returns

Returns nil.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
rc = api.sqlany_free_stmt( stmt )
```

sqlany_get_bind_param_info Function

Retrieves information about the parameters that were bound using `sqlany_bind_param`.

Syntax

```
sqlany_get_bind_param_info ( $stmt, $index )
```

Parameters

- **\$stmt** – A statement successfully prepared using `sqlany_prepare`.
- **\$index** – The index of the parameter. The number must be between 0 and `sqlany_num_params() - 1`.

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and a described parameter as the second element.

Example

```
# Get information on first parameter (0)
rc, param_info = api.sqlany_get_bind_param_info( stmt, 0 )
print "Param_info direction = ", param_info.get_direction(), "\n"
print "Param_info output = ", param_info.get_output(), "\n"
```

sqlany_get_column Function

Returns the value fetched for the specified column.

Syntax

```
sqlany_get_column ( $stmt, $col_index )
```

Parameters

- **\$stmt** – A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.
- **\$col_index** – The number of the column to be retrieved. A column number is between 0 and `sqlany_num_cols()` - 1.

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and the column value as the second element.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_get_column_info Function

Gets column information for the specified result set column.

Syntax

```
sqlany_get_column_info ( $stmt, $col_index )
```

Parameters

- **\$stmt** – A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.
- **\$col_index** – The column number between 0 and `sqlany_num_cols()` - 1.

Returns

Returns a 9-element array of information describing a column in a result set. The first element contains 1 on success or 0 on failure. The array elements are described in the following table.

Element number	Type	Description
0	Integer	1 on success or 0 on failure.
1	Integer	Column index (0 to sqlany_num_cols() - 1).
2	String	Column name.
3	Integer	Column type.
4	Integer	Column native type.
5	Integer	Column precision (for numeric types).
6	Integer	Column scale (for numeric types).
7	Integer	Column size.
8	Integer	Column nullable (1=nullable, 0=not nullable).

Example

```
# Get column info for first column (0)
rc, col_num, col_name, col_type, col_native_type, col_precision,
col_scale,
  col_size, col_nullable = api.sqlany_get_column_info( stmt, 0 )
```

sqlany_get_next_result Function

Advances to the next result set in a multiple result set query.

Syntax

```
sqlany_get_next_result ( $stmt )
```

Parameters

- **\$stmt** – A statement object executed by `sqlany_execute` or `sqlany_execute_direct`.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
stmt = api.sqlany_prepare(conn, "call two_results()" )
rc = api.sqlany_execute( stmt )
# Fetch from first result set
rc = api.sqlany_fetch_absolute( stmt, 3 )
# Go to next result set
rc = api.sqlany_get_next_result( stmt )
```

```
# Fetch from second result set
rc = api.sqlany_fetch_absolute( stmt, 2 )
```

sqlany_init Function

Initializes the interface.

Syntax

```
sqlany_init ( )
```

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and the Ruby interface version as the second element.

Example

```
# Load the SQLAnywhere gem
begin
  require 'rubygems'
  gem 'sqlanywhere'
  unless defined? SQLAnywhere
    require 'sqlanywhere'
  end
end
# Create an interface
api = SQLAnywhere::SQLAnywhereInterface.new()
# Initialize the interface (loads the DLL/SO)
SQLAnywhere::API.sqlany_initialize_interface( api )
# Initialize our api object
api.sqlany_init()
```

sqlany_new_connection Function

Creates a connection object.

Syntax

```
sqlany_new_connection ( )
```

Returns

Returns a scalar value that is a connection object.

Remarks

A connection object must be created before a database connection is established. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time.

Example

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
```

```
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Status=#{status}\n"
```

sqlany_num_cols Function

Returns number of columns in the result set.

Syntax

```
sqlany_num_cols ( $stmt )
```

Parameters

- **\$stmt** – A statement object executed by `sqlany_execute` or `sqlany_execute_direct`.

Returns

Returns a scalar value that is the number of columns in the result set, or -1 on a failure.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of result set columns
num_cols = api.sqlany_num_cols( stmt )
```

sqlany_num_params Function

Returns the number of parameters that are expected for a prepared statement.

Syntax

```
sqlany_num_params ( $stmt )
```

Parameters

- **\$stmt** – A statement object returned by the successful execution of `sqlany_prepare`.

Returns

Returns a scalar value that is the number of parameters in a prepared statement, or -1 on a failure.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
num_params = api.sqlany_num_params( stmt )
```

sqlany_num_rows Function

Returns the number of rows in the result set.

Syntax

```
sqlany_num_rows ( $stmt )
```

Parameters

- **\$stmt** – A statement object executed by `sqlany_execute` or `sqlany_execute_direct`.

Returns

Returns a scalar value that is the number of rows in the result set. If the number of rows is an estimate, the number returned is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

Remarks

By default, this function only returns an estimate. To return an exact count, set the `ROW_COUNTS` option on the connection.

A count of the number of rows in a result set can be returned only for the first result set in a statement that returns multiple result sets. If `sqlany_get_next_result` is used to move to the next result set, `sqlany_num_rows` will still return the number of rows in the first result set.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of rows in result set
num_rows = api.sqlany_num_rows( stmt )
```

sqlany_prepare Function

Prepares the supplied SQL string.

Syntax

```
sqlany_prepare ( $conn, $sql )
```

Parameters

- **\$conn** – A connection object with a connection established using `sqlany_connect`.
- **\$sql** – The SQL statement to be prepared.

Returns

Returns a scalar value that is the statement object, or nil on failure.

Remarks

The statement associated with the statement object is executed by `sqlany_execute`. You can use `sqlany_free_stmt` to free the resources associated with the statement object.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
```

```
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_rollback Function

Rolls back the current transaction.

Syntax

```
sqlany_rollback ( $conn )
```

Parameters

- **\$conn** – The connection object on which the rollback operation is to be performed.

Returns

Returns a scalar value that is 1 on success, 0 on failure.

Example

```
rc = api.sqlany_rollback( conn )
```

sqlany_sqlstate Function

Retrieves the current SQLSTATE.

Syntax

```
sqlany_sqlstate ( $conn )
```

Parameters

- **\$conn** – A connection object returned from `sqlany_new_connection`.

Returns

Returns a scalar value that is the current five-character SQLSTATE.

Example

```
sql_state = api.sqlany_sqlstate( conn )
```

Column Types

The following Ruby class defines the column types returned by some SQL Anywhere Ruby functions.

```
class Types
  A_INVALID_TYPE = 0
  A_BINARY       = 1
  A_STRING       = 2
  A_DOUBLE       = 3
  A_VAL64        = 4
  A_UVAL64       = 5
  A_VAL32        = 6
  A_UVAL32       = 7
end
```



```

A_VAL16      = 8
A_UVAL16    = 9
A_VAL8       = 10
A_UVAL8     = 11
end

```

Native Column Types

The following table defines the native column types returned by some SQL Anywhere functions.

Native type value	Native type
384	DT_DATE
388	DT_TIME
390	DT_TIMESTAMP_STRUCT
392	DT_TIMESTAMP
448	DT_VARCHAR
452	DT_FIXCHAR
456	DT_LONGVARCHAR
460	DT_STRING
480	DT_DOUBLE
482	DT_FLOAT
484	DT_DECIMAL
496	DT_INT
500	DT_SMALLINT
524	DT_BINARY
528	DT_LONGBINARY
600	DT_VARIABLE
604	DT_TINYINT
608	DT_BIGINT
612	DT_UNSENT
616	DT_UNSSMALLINT
620	DT_UNSBIGINT

Ruby Support

Native type value	Native type
624	DT_BIT
628	DT_NSTRING
632	DT_NFIXCHAR
636	DT_NVARCHAR
640	DT_LONGNVARCHAR

Sybase Open Client Support

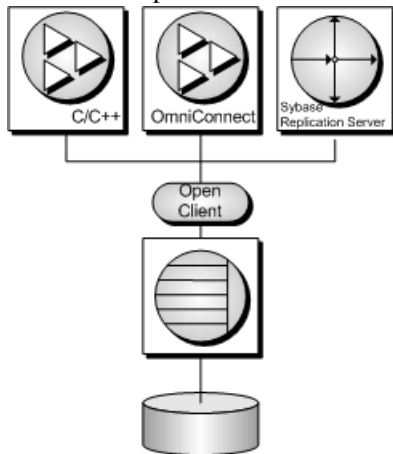
Sybase Open Client provides customer applications, third-party products, and other Sybase products with the interfaces needed to communicate with SAP Sybase IQ and other Open Servers.

When to use Open Client

You should consider using the Open Client interface if you are concerned with Adaptive Server Enterprise compatibility or if you are using other Sybase products that support the Open Client interface.

Open Client applications

You can develop applications in C or C++, and then connect those applications to SAP Sybase IQ using the Open Client API. Other Sybase applications, such as OmniConnect, use Open Client. The Open Client API is also supported by Sybase Adaptive Server Enterprise.



Open Client Architecture

This section describes the Open Client programming interface for SAP Sybase IQ. The primary documentation for Sybase Open Client application development is the Sybase Open Client documentation, available from SAP. This section describes features specific to SAP Sybase IQ, but it is not an exhaustive guide to Sybase Open Client application programming.

Sybase Open Client has two components: programming interfaces and network services.

DB-Library and Client Library

Sybase Open Client provides two core programming interfaces for writing client applications: DB-Library and Client-Library.

Open Client DB-Library provides support for older Open Client applications, and is a completely separate programming interface from Client-Library. DB-Library is documented in the *Open Client DB-Library/C Reference Manual*, provided with the Sybase Open Client product.

Client-Library programs also depend on CS-Library, which provides routines that are used in both Client-Library and Server-Library applications. Client-Library applications can also use routines from Bulk-Library to help high-speed data transfer.

Both CS-Library and Bulk-Library are included in the Sybase Open Client, which is available separately.

Network services

Open Client network services include Sybase Net-Library, which provides support for specific network protocols such as TCP/IP and DECnet. The Net-Library interface is invisible to application developers. However, on some platforms, an application may need a different Net-Library driver for different system network configurations. Depending on your host platform, the Net-Library driver is specified either by the system's Sybase configuration or when you compile and link your programs.

Instructions for driver configuration can be found in the *Open Client/Server Configuration Guide*.

Instructions for building Client-Library programs can be found in the *Open Client/Server Programmer's Supplement*.

What You Need to Build Open Client Applications

To run Open Client applications, you must install and configure Sybase Open Client components on the computer where the application is running. You may have these components present as part of your installation of other Sybase products or you can optionally install these libraries with SAP Sybase IQ, subject to the terms of your license agreement.

Open Client applications do not need any Open Client components on the computer where the database server is running.

To build Open Client applications, you need the development version of Open Client, available from SAP.

By default, SAP Sybase IQ databases are created as case-insensitive, while Adaptive Server databases are case-sensitive.

Open Client Data Type Mappings

Sybase Open Client has its own internal data types, which differ in some details from those available in SAP Sybase IQ. For this reason, SAP Sybase IQ internally maps some data types between those used by Open Client applications and those available in SAP Sybase IQ.

To build Open Client applications, you need the development version of Open Client. To use Open Client applications, the Open Client run-times must be installed and configured on the computer where the application runs.

The SAP Sybase IQ server does not require any external communications runtime to support Open Client applications.

Each Open Client data type is mapped onto the equivalent SAP Sybase IQ data type. All Open Client data types are supported.

SAP Sybase IQ Data Types with no Direct Counterpart in Open Client

The following table lists the mappings of data types supported in SAP Sybase IQ that have no direct counterpart in Open Client.

SAP Sybase IQ data type	Open Client data type
unsigned short	int
unsigned int	bigint
unsigned bigint	numeric(20,0)
string	varchar
timestamp	datetime

Range Limitations in Open Client Data Type Mapping

Some data types have different ranges in SAP Sybase IQ than in Open Client. In such cases, overflow errors can occur during retrieval or insertion of data.

The following table lists Open Client application data types that can be mapped to SAP Sybase IQ data types, but with some restriction in the range of possible values.

The Open Client data type is usually mapped to an SAP Sybase IQ data type with a greater range of possible values. As a result, it is possible to pass a value to SAP Sybase IQ that will be accepted and stored in a database, but that is too large to be fetched by an Open Client application.

Data type	Open Client lower range	Open Client upper range	SAP Sybase IQ lower range	SAP Sybase IQ upper range
MONEY	-922 377 203 685 477.5808	922 377 203 685 477.5807	-999 999 999 999 999.9999	999 999 999 999 999.9999
SMALLMONEY	-214 748.3648	214 748.3647	-999 999.9999	-999 999.9999
DATETIME [1]	January 1, 1753	December 31, 9999	January 1, 0001	December 31, 9999
SMALLDATE-TIME	January 1, 1900	June 6, 2079	January 1, 0001	December 31, 9999

[1] For versions earlier than OpenClient 15.5; otherwise, the full range of dates from 0001-01-01 to 9999-12-31 is supported.

Example

For example, the Open Client MONEY and SMALLMONEY data types do not span the entire numeric range of their underlying SAP Sybase IQ implementations. Therefore, it is possible to have a value in an SAP Sybase IQ column which exceeds the boundaries of the Open Client data type MONEY. When the client fetches any such offending values via SAP Sybase IQ, an error is generated.

Timestamps

TIMESTAMP values inserted into or retrieved from SAP Sybase IQ will have the date portion restricted to January 1, 1753 or later and the time version restricted to 1/300th of a second precision if the client is using Open Client 15.1 or earlier. If, however, the client is using Open Client 15.5 or later, then no restriction will apply to TIMESTAMP values.

SQL in Open Client Applications

This section provides a very brief introduction to using SQL in Open Client applications, with a particular focus on SAP Sybase IQ-specific issues.

For a complete description, see the Open Client documentation at <http://www.sybase.com/products/databasemanagement/openserver>.

Open Client SQL Statement Execution

You send SQL statements to a database server by including them in Client Library function calls. For example, the following pair of calls executes a DELETE statement:

```
ret = ct_command(cmd, CS_LANG_CMD,
                "DELETE FROM Employees
                WHERE EmployeeID=105"
                CS_NULLTERM,
```

```

        CS_UNUSED) ;
ret = ct_send(cmd) ;

```

Open Client Prepared Statements

The `ct_dynamic` function is used to manage prepared statements. This function takes a *type* parameter that describes the action you are taking.

Perform the following tasks to use a prepared statement in Open Client:

1. Prepare the statement using the `ct_dynamic` function, with a `CS_PREPARE` *type* parameter.
2. Set statement parameters using `ct_param`.
3. Execute the statement using `ct_dynamic` with a `CS_EXECUTE` *type* parameter.
4. Free the resources associated with the statement using `ct_dynamic` with a `CS_DEALLOC` *type* parameter.

For more information about using prepared statements in Open Client, see your Open Client documentation.

Open Client Cursor Management

The `ct_cursor` function is used to manage cursors. This function takes a *type* parameter that describes the action you are taking.

Supported Cursor Types

Not all the types of cursor that SAP Sybase IQ supports are available through the Open Client interface. You cannot use scroll cursors, dynamic scroll cursors, or insensitive cursors through Open Client.

Uniqueness and updatability are two properties of cursors. Cursors can be unique (each row carries primary key or uniqueness information, regardless of whether it is used by the application) or not. Cursors can be read-only or updatable. If a cursor is updatable and not unique, performance may suffer, as no prefetching of rows is done in this case, regardless of the `CS_CURSOR_ROWS` setting.

The Steps in Using Cursors

In contrast to some other interfaces, such as embedded SQL, Open Client associates a cursor with a SQL statement expressed as a string. Embedded SQL first prepares a statement and then the cursor is declared using the statement handle.

Perform the following tasks to use cursors in Open Client:

1. To declare a cursor in Open Client, use `ct_cursor` with `CS_CURSOR_DECLARE` as the *type* parameter.
2. After declaring a cursor, you can control how many rows are prefetched to the client side each time a row is fetched from the server by using `ct_cursor` with `CS_CURSOR_ROWS` as the *type* parameter.

Storing prefetched rows at the client side reduces the number of calls to the server and this improves overall throughput and turnaround time. Prefetched rows are not immediately passed on to the application; they are stored in a buffer at the client side ready for use.

The setting of the prefetch database option controls prefetching of rows for other interfaces. It is ignored by Open Client connections. The CS_CURSOR_ROWS setting is ignored for non-unique, updatable cursors.

3. To open a cursor in Open Client, use `ct_cursor` with CS_CURSOR_OPEN as the *type* parameter.
4. To fetch each row in to the application, use `ct_fetch`.
5. To close a cursor, you use `ct_cursor` with CS_CURSOR_CLOSE.
6. In Open Client, you also need to deallocate the resources associated with a cursor. You do this by using `ct_cursor` with CS_CURSOR_DEALLOC. You can also use CS_CURSOR_CLOSE with the additional parameter CS_DEALLOC to perform these operations in a single step.

Open Client Row Modification Through a Cursor

With Open Client, you can delete or update rows in a cursor, as long as the cursor is for a single table. The user must have permissions to update the table and the cursor must be marked for update.

Instead of carrying out a fetch, you can delete or update the current row of the cursor using `ct_cursor` with CS_CURSOR_DELETE or CS_CURSOR_UPDATE, respectively.

You cannot insert rows through a cursor in Open Client applications.

Open Client Result Sets

Open Client handles result sets in a different way than some other SAP Sybase IQ interfaces.

In embedded SQL and ODBC, you *describe* a query or stored procedure to set up the proper number and types of variables to receive the results. The description is done on the statement itself.

In Open Client, you do not need to describe a statement. Instead, each row returned from the server can carry a description of its contents. If you use `ct_command` and `ct_send` to execute statements, you can use the `ct_results` function to handle all aspects of rows returned in queries.

If you do not want to use this row-by-row method of handling result sets, you can use `ct_dynamic` to prepare a SQL statement and use `ct_describe` to describe its result set. This corresponds more closely to the describing of SQL statements in other interfaces.

Known Open Client Limitations of SAP Sybase IQ

Using the Open Client interface, you can use an SAP Sybase IQ database in much the same way as you would an Adaptive Server Enterprise database. There are some limitations, including the following:

- SAP Sybase IQ does not support the Adaptive Server Enterprise Commit Service.
- A client/server connection's *capabilities* determine the types of client requests and server responses permitted for that connection. The following capabilities are not supported:

- CS_CSR_ABS
 - CS_CSR_FIRST
 - CS_CSR_LAST
 - CS_CSR_PREV
 - CS_CSR_REL
 - CS_DATA_BOUNDARY
 - CS_DATA_SENSITIVITY
 - CS_OPT_FORMATONLY
 - CS_PROTO_DYNPROC
 - CS_REG_NOTIF
 - CS_REQ_BCP

- Security options, such as SSL, are not supported. However, password encryption is supported.
- Open Client applications can connect to SAP Sybase IQ using TCP/IP.
For more information about capabilities, see the *Open Server Server-Library C Reference Manual*.
- When the CS_DATAFMT is used with the CS_DESCRIBE_INPUT, it does not return the data type of a column when a parameterized variable is sent to SAP Sybase IQ as input.

HTTP Web Services

Develop web service applications that use SAP Sybase IQ as an HTTP web server.

SAP Sybase IQ As an HTTP Web Server

SAP Sybase IQ contains a built-in HTTP web server that allows you to create online web services in SAP Sybase IQ databases. SAP Sybase IQ web servers support HTTP and SOAP over HTTP requests sent by web browsers and client applications. The web server performance is optimized because web services are embedded in the database.

SAP Sybase IQ web services provide client applications with an alternative to traditional interfaces, such as JDBC and ODBC. They are easily deployed because additional components are not needed, and can be accessed from multi-platform client applications written in a variety of languages, including scripting languages—such as Perl and Python.

In addition to providing web services over an HTTP web server, SAP Sybase IQ can function as a SOAP or HTTP client application to access standard web services available over the Internet and other SAP Sybase IQ HTTP web servers.

Quick Start to Using SAP Sybase IQ As an HTTP Web Server

This section illustrates how to start an SAP Sybase IQ HTTP web server, create a web service, and access it from a web browser. It does not illustrate SAP Sybase IQ web service features, such as SOAP over HTTP support and application development, to a full extent. Many SAP Sybase IQ web service features are available that are beyond the scope of this guide.

Perform the following tasks to create an SAP Sybase IQ HTTP web server and HTTP web service:

1. Start the SAP Sybase IQ HTTP web server while loading an SAP Sybase IQ database. Run the following command at a command prompt:

```
iqsrsv16 -xs http(port=8082) iqdemo.db
```

Note: Use the `iqsrsv16` command to start a database server that can be accessed on a network.

The `-xs http(port=8082)` option instructs the server to listen for HTTP requests on port 8082. Use a different port number if a web server is already running on port 8082.

2. Use the `CREATE SERVICE` statement to create a web service that responds to incoming web browser requests.
 - a. Connect to the `demo.db` database using Interactive SQL by running the following command:

```
dbisql -c "dbf=iqdemo.db;uid=<user_id>;pwd=<password>"
```

- b.** Create a new web service in the database.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE SampleWebService  
  TYPE 'web-service-type-clause'  
  AUTHORIZATION OFF  
  USER DBA  
  AS SELECT 'Hello world!';
```

Replace *web-service-type-clause* with the desired web service type. The HTML type clause is recommended for web browser compatibility. Other general HTTP web service type clauses include XML, RAW, and JSON.

The CREATE SERVICE statement creates the SampleWebService web service, which returns the result set of the SELECT statement. In this example, the statement returns "Hello world!"

The AUTHORIZATION OFF clause indicates that authorization is not required to access the web service.

The USER DBA statement indicates that the service statement should be run under the DBA login name.

The AS SELECT clause allows the service to select from a table or function, or view data directly. Use AS CALL as an alternative clause to call a stored procedure.

- 3.** View the web service in a web browser.

On the computer running the SAP Sybase IQ HTTP web server, open a web browser, such as Internet Explorer or Firefox, and go to the following URL:

```
http://localhost:8082/demo/SampleWebService
```

This URL directs your web browser to the HTTP web server on port 8082.

SampleWebService prints "Hello world". The result set output is displayed in the format specified by the *web-service-type-clause* from step 2.

Other Sample Resources

Samples are included in the %ALLUSERSPROFILE%\SybaseIQ\samples \SQLAnywhere\http directory.

Other examples might be available on CodeXchange at <http://www.sybase.com/developer/codexchange>.

How to Start an HTTP Web Server

The SAP Sybase IQ HTTP web server starts automatically when you launch the database server with the -xs server option. This option allows you to perform the following tasks:

- Enable a web service protocol to listen for web service requests.

- Configure network protocol options, such as server port, logging, time-out criteria, and the maximum request size.

The general format of the command line is as follows:

```
iqsrvt16 -xs protocol-type(protocol-options) your-database-name.db
```

Replace *protocol-type* and *protocol-options* with one of the following supported protocols and any appropriate protocol options:

- **HTTP** – Use this protocol to listen for HTTP connections. Here is an example.

```
iqsrvt16 -xs HTTP(PORT=8082) services.db
```

- **HTTPS** – Use this protocol to listen for HTTPS connections. SSL version 3.0 and TLS version 1.0/1.1 are supported. Here is an example.

```
iqsrvt16 -xs "HTTPS (FIPS=N;PORT=8082;IDENTITY="%ALLUSERSPROFILE
%\SybaseIQ\samples\Certificates
\rsaserver.id;IDENTITY_PASSWORD=test)" services.db
```

Note: Network protocol options are available for each supported protocol. These options allow you to control protocol behavior and can be configured at the command line when you launch your database server.

Configuration of Network Protocol Options

Network protocol options are optional settings that provide control over a specified web service protocol. These settings are configured at the command line when you launch your database server with the `-xs` database server option.

For example, the following command line configures an HTTPS listener with the `PORT`, `FIPS`, `Identity`, and `Identity_Password` network protocol options specified:

```
iqsrvt16 -xs https (PORT=544;FIPS=YES;
IDENTITY=certificate.id;IDENTITY_PASSWORD=password) your-
database-name.db
```

This command starts a database server that enables the HTTPS web service protocol for the `your-database-name.db` database. The network protocol options indicate that the web server should perform the following tasks:

- Listen on port 544 instead of the default HTTPS port (443).
- Enable FIPS-approved security algorithms to encrypt communications.
- Locate the specified identity file, `certificate.id`, which contains a public certificate and its private key.
- Validate the private key against the specified identity password, `password`.

The following list identifies the network protocol options that are commonly used for web service protocols:

Network protocol option	Available web service protocols	Description
DatabaseName (DBN) protocol option	HTTP, HTTPS	Specifies the name of a database to use when processing web requests, or uses the REQUIRED or AUTO keyword to specify whether database names are required as part of the URL.
FIPS protocol option	HTTPS	Enables FIPS-approved security algorithms to encrypt database files, communications for database client/server communication, and web services.
Identity protocol option	HTTPS	Specifies the name of an identity file to use for secure HTTPS connections.
Identity_Password protocol option	HTTPS	Specifies the password for the encryption certificate.
LocalOnly (LO) protocol option	HTTP, HTTPS	Allows a client to choose to connect only to a server on the local computer, if one exists.
LogFile (LOG) protocol option	HTTP, HTTPS	Specifies the name of the file where the database server writes information about web requests.
LogFormat (LF) protocol option	HTTP, HTTPS	Controls the format of messages written to the log file where the database server writes information about web requests, and specifies which fields appear in the messages.
LogOptions (LOPT) protocol option	HTTP, HTTPS	Specifies the types of messages that are recorded in the log where the database server writes information about web requests.
ServerPort (PORT) protocol option	HTTP, HTTPS	Specifies the port on which the database server is listening.

How to Start Multiple HTTP Web Servers

A multiple HTTP web server configuration allows you to create web services across databases and have them appear as part of a single web site. You can start multiple HTTP web servers by

using multiple instances of the `-xs` database server option. This task is performed by specifying a unique port number for each HTTP web server.

Example

In this example, the following command line starts two HTTP web services—one for `your-first-database.db` and one for `your-second-database.db`:

```
iqsrv16 -xs http(port=80;dbn=your-first-database),http(port=8800;dbn=your-second-database)
your-first-database.db your-second-database.db
```

What Are Web Services

Web services refer to software that assists inter-computer data transfer and interoperability. They make segments of business logic available over the Internet. URLs become available to clients when managing web services in an HTTP web server. The conventions used when specifying a URL determine how the server should communicate with web clients.

Web service management involves the following tasks:

- Choosing the types of web services that you want to manage.
- Creating and maintaining those web services

Web services can be created and stored in a SQL Anywhere database.

Web Service Types

When a web browser or client application makes a web service request to an SAP Sybase IQ web service, the request is processed and a result set is returned in the response. SAP Sybase IQ supports several web service types that provide control over the result set format and how result sets are returned. You specify the web server type with the `TYPE` clause of the `CREATE SERVICE` or `ALTER SERVICE` statement after choosing an appropriate web service type.

The following web service types are supported:

- **HTML** – The result set of a statement, function, or procedure is formatted into an HTML document that contains a table. Web browsers display the body of the HTML document.
- **XML** – The result set of a statement, function, or procedure is returned as an XML document. Non-XML formatted result sets are automatically formatted into XML. Web browsers display the raw XML code, including tags and attributes.

The XML formatting is the equivalent of using the `FOR XML RAW` clause in a `SELECT` statement, such as in the following SQL statement example:

```
SELECT * FROM table-name FOR XML RAW
```

- **RAW** – The result set of a statement, function, or procedure is returned without automatic formatting.

This service type provides the most control over the result set. However, you must generate the response by writing the necessary markup (HTML, XML) explicitly within your stored procedure. You can use the `SA_SET_HTTP_HEADER` system procedure to set the HTTP

Content-Type header to specify the MIME type, allowing web browsers to correctly display the result set.

- **JSON** – The result set of a statement, function, or procedure is returned in JSON (JavaScript Object Notation). JavaScript Object Notation (JSON) is a language-independent, text-based data interchange format developed for the serialization of JavaScript data. JSON represents four basic types: strings, numbers, booleans, and NULL. JSON also represents two structured types: objects and arrays. For more information about JSON, see <http://www.json.org/>.

This service is used by AJAX to make HTTP calls to web applications. For an example of the JSON type, see `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\json_sample.sql`.

- **SOAP** – The result set of a statement, function, or procedure is returned as a SOAP response. SOAP services provide a common data interchange standard to provide data access to disparate client applications that support SOAP. SOAP request and response envelopes are transported as an XML payload using HTTP (SOAP over HTTP). A request to a SOAP service must be a valid SOAP request, not a general HTTP request. The output of SOAP services can be adjusted using the `FORMAT` and `DATATYPE` attributes of the `CREATE` or `ALTER SERVICE` statement.
- **DISH** – A DISH service (Determine SOAP Handler) is an SAP Sybase IQ SOAP endpoint. The DISH service exposes the WSDL (Web Services Description Language) document that describes all SOAP Operations (SAP Sybase IQ SOAP services) accessible through it. A SOAP client toolkit builds the client application with interfaces based on the WSDL. The SOAP client application directs all SOAP requests to the SOAP endpoint (the SAP Sybase IQ DISH service).

Example

The following example illustrates the creation of a general HTTP web service that uses the RAW service type:

```
CREATE PROCEDURE sp_echotext(str LONG VARCHAR)
BEGIN
    CALL sa_set_http_header( 'Content-Type', 'text/plain' );
    SELECT str;
END;

CREATE SERVICE SampleWebService
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL sp_echotext ( :str );
```


Web Service Maintenance

Web service maintenance involves the following tasks:

- **Creating or altering web services** – Create or alter web services to provide web applications supporting a web browser interface and provide data interchange over the web using REST and SOAP methodologies.
- **Dropping web services** – Dropping a web service causes the subsequent requests made for that service to return a 404 Not Found HTTP status message. All unresolved requests, intended or unintended, are processed if a root web service exists.
- **Commenting on web services** – Commenting is optional and allows you to provide documentation for your web services.
- **Creating and customizing a root web service** – You can create a root web service to handle HTTP requests that do not match any other web service requests.
- **Enabling and disabling web services** – A disabled web service returns a 404 Not Found HTTP status message. The METHOD clause specifies the HTTP methods that can be called for a particular web service.

How to Create or Alter a Web Service

Creating or altering a web service requires use of the CREATE SERVICE or ALTER SERVICE statement, respectively. This section illustrates how to execute these statements with Interactive SQL to create different kinds of web services. The examples in this section assume that you have connected to an SAP Sybase IQ database, *your-database.db*, through Interactive SQL using the following command:

```
dbisql -c "dbf=your-database.db;uid=your-userid;pwd=your-password"
```

How to Create HTTP Web Services

HTTP web services are classified as HTML, XML or RAW. All HTTP web services can be created or altered using the same CREATE SERVICE and ALTER SERVICE statement syntax.

Example

Execute the following statement in Interactive SQL to create a sample general HTTP web service in the HTTP web server:

```
CREATE SERVICE SampleWebService
  TYPE 'web-service-type-clause'
  URL OFF
  USER DBA
  AUTHORIZATION OFF
  AS sql-statement;
```

The CREATE SERVICE statement creates a new web service named `SampleWebService` and returns the result set of *sql-statement*. You can replace *sql-statement* with either a SELECT statement to select data from a table or view directly, or a CALL statement to call a stored procedure in the database.

Replace *web-service-type-clause* with the desired web service type. Valid clauses for HTTP web services include HTML, XML, RAW and JSON.

You can view the generated result set for the `SampleWebService` service by accessing the service in a web browser.

How to Create SOAP Over HTTP Services

SOAP is a data interchange standard supported by many development environments.

A SOAP payload consists of an XML document, known as a SOAP envelope. A SOAP request envelope contains the SOAP operation (a SOAP service) and specifies all appropriate parameters. The SOAP service parses the request envelope to obtain the parameters and calls or selects a stored procedure or function just as any other service does. The presentation layer of the SOAP service streams the result set back to the client within a SOAP envelope in a predefined format as specified by the DISH service's WSDL. For more information about SOAP standards, see <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

By default, SOAP service parameters and result data are typed as XmlSchema string parameters. DATATYPE ON specifies that input parameters and response data should use TRUE types. Specifying DATATYPE changes the WSDL specification accordingly, so that client SOAP toolkits generate interfaces with the appropriate type of parameters and response objects.

The FORMAT clause is used to target specific SOAP toolkits with varying capabilities. DNET provides Microsoft .NET client applications to consume a SOAP service response as a System.Data.DataSet object. CONCRETE exposes a more general structure that allows an object-oriented application, such as .NET or Java, to generate response objects that package rows and columns. XML returns the entire response as an XML document, exposing it as a string. Clients can further process the data using an XML parser. The FORMAT clause of the CREATE SERVICE statement supports multiple client application types.

Note: The DATATYPE clause only pertains to SOAP services (there is no data typing in HTML) The FORMAT clause can be specified for either a SOAP or DISH service. A SOAP service FORMAT specification overrides that of the DISH service.

Example

Execute the following statement in Interactive SQL to create a SOAP over HTTP service:

```
CREATE SERVICE SampleSOAPService
  TYPE 'SOAP'
  DATATYPE ON
  FORMAT 'CONCRETE'
  USER DBA
  AUTHORIZATION OFF
  AS sql-statement;
```

How to Create DISH Services

SAP Sybase IQ allows you to create DISH services that act as SOAP endpoints for groups of SOAP services. DISH services also automatically construct WSDL (Web Services Description Language) documents that allow SOAP client toolkits to generate the interfaces necessary to interchange data with the SOAP services described by the WSDL.

SOAP services can be added and removed without requiring maintenance to the DISH services because the current working set of SOAP over HTTP services are always exposed.

Example

Execute the following SQL statements in Interactive SQL to create sample SOAP and DISH services in the HTTP web server:

```
CREATE SERVICE "Samples/TestSoapOp"
  TYPE 'SOAP'
  DATATYPE ON
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT( ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR )
BEGIN
  SELECT i, f, s;
END;

CREATE SERVICE "dnet_endpoint"
  TYPE 'DISH'
  GROUP "Samples"
  FORMAT 'DNET';
```

The first `CREATE SERVICE` statement creates a new SOAP service named `Samples/TestSoapOp`.

The second `CREATE SERVICE` statement creates a new DISH service named `dnet_endpoint`. The `Samples` portion of the `GROUP` clause identifies the group of SOAP services to expose. You can view the WSDL document generated by the DISH service. When running your SAP Sybase IQ web server on a computer, you can access the service using the `http://localhost:port-number/dnet_endpoint` URL, where *port-number* is the port number that the server is running on.

In this example, the SOAP service does not contain a `FORMAT` clause to indicate a SOAP response format. Therefore, the SOAP response format is dictated by the DISH service, which does not override the `FORMAT` clause of the SOAP service. This feature allows you to create homogeneous DISH services where each DISH endpoint can serve SOAP clients with varying capabilities.

Creating homogeneous DISH services

When SOAP service definitions defer the specification of the `FORMAT` clause to the `DISH` service, a set of SOAP services can be grouped together within a `DISH` service that defines the format. Multiple `DISH` services can then expose the same group of SOAP services with a different `FORMAT` specifications. If you expand on the `TestSoapOp` example, you can create another `DISH` service named `java_endpoint` using the following SQL statement:

```
CREATE SERVICE "java_endpoint"  
  TYPE 'DISH'  
  GROUP "Samples"  
  FORMAT 'CONCRETE';
```

In this example, the SOAP client receives a response object named `TestSoapOp_Dataset` when it makes a web service request for the `TestSoapOp` operation through the `java_endpoint` `DISH` service. The WSDL can be inspected to compare the differences between `dnet_endpoint` and `java_endpoint`. Using this technique, a SOAP endpoint can quickly be constructed to meet the needs of a particular SOAP client toolkit.

How to Drop a Web Service

Dropping a web service causes the subsequent requests made for that service to return a 404 Not Found HTTP status message. All unresolved requests, intended or unintended, are processed if a `root` web service exists.

Example

Execute the following SQL statement to drop a web service named `SampleWebService`:

```
DROP SERVICE SampleWebService;
```

How to Comment a Web Service

Providing documentation for a web service requires use of the `COMMENT ON SERVICE` statement. A comment can be removed by setting the `statement` clause to null.

Example

For example, execute the following SQL statement to create a new comment on a web service named `SampleWebService`:

```
COMMENT ON SERVICE SampleWebService  
  IS "This is a comment on my web service.";
```

How to Create and Customize a Root Web Service

An HTTP client request that does not match any web service request is processed by the `root` web service if a `root` web service is defined.

The `root` web service provides you with an easy and flexible method to handle arbitrary HTTP requests whose URLs are not necessarily known at the time when you build your application, and to handle unrecognized requests.

Example

This example illustrates how to use a `root` web service, which is stored in a table within the database, to provide content to web browsers and other HTTP clients. It assumes that you have started a local HTTP web server on a single database and listening on port 80. All scripts are run on the web server.

Connect to the database server through Interactive SQL and execute the following SQL statement to create a `root` web service that passes the `url` host variable, which is supplied by the client, to a procedure named `PageContent`:

```
CREATE SERVICE root
  TYPE 'RAW'
  AUTHORIZATION OFF
  SECURE OFF
  URL ON
  USER DBA
  AS CALL PageContent(:url);
```

The `URL ON` portion specifies that the full path component is made accessible by an HTTP variable named `URL`.

Execute the following SQL statement to create a table for storing page content. In this example, the page content is defined by its `URL`, `MIME`-type, and the content itself.

```
CREATE TABLE Page_Content (
  url          VARCHAR(1024) NOT NULL PRIMARY KEY,
  content_type VARCHAR(128)  NOT NULL,
  image       LONG VARCHAR  NOT NULL
);
```

Execute the following SQL statements to populate the table. In this example, the intent is to define the content to be provided to the HTTP client when the `index.html` page is requested.

```
INSERT INTO Page_Content
VALUES (
  'index.html',
  'text/html',
  '<html><body><h1>Hello World</h1></body></html>'
);
COMMIT;
```

Execute the following SQL statements to implement the `PageContent` procedure, which accepts the `url` host variable that is passed through to the `root` web service:

```

CREATE PROCEDURE PageContent(IN @url LONG VARCHAR)
RESULT ( html_doc LONG VARCHAR )
BEGIN
    DECLARE @status CHAR(3);
    DECLARE @type VARCHAR(128);
    DECLARE @image LONG VARCHAR;

    SELECT content_type, image INTO @type, @image
    FROM Page_Content
    WHERE url = @url;

    IF @image is NULL THEN
        SET @status = '404';
        SET @type = 'text/html';
        SET @image = '<html><body><h1>404 - Page Not Found</h1>'
            || '<p>There is no content located at the URL "'
            || html_encode( @url ) || '" on this server.<p>'
            || '</body></html>';
    ELSE
        SET @status = '200';
    END IF;
    CALL sa_set_http_header( '@HttpStatus', @status );
    CALL sa_set_http_header( 'Content-Type', @type );
    SELECT @image;
END;

```

The root web service calls the PageContent procedure when a request to the HTTP server does not match any other defined web service URL. The procedure checks if the client-supplied URL matches a url in the Page_Content table. The SELECT statement sends a response to the client. If the client-supplied URL was not found in the table, a generic 404 - Page Not Found html page is built and sent to the client.

Some browsers will respond to the 404 status with their own page, so there is no guarantee that the generic page will be displayed.

In the error message, the HTML_ENCODE function is used to encode the special characters in the client-supplied URL.

The @HttpStatus header is used to set the status code returned with the request. A 404 status indicates a Not Found error, and a 200 status indicates OK. The 'Content-Type' header is used to set the content type returned with the request. In this example, the content (MIME) type of the index.html page is text/html.

Web Service SQL Statements

The following SQL statements are available to assist with web service development:

Web server related SQL statements	Description
CREATE SERVICE statement [HTTP web service]	Creates a new HTTP web service.

Web server related SQL statements	Description
CREATE SERVICE statement [SOAP web service]	Creates a new SOAP over HTTP or DISH service.
ALTER SERVICE statement [HTTP web service]	Alters an existing HTTP web service.
ALTER SERVICE statement [SOAP web service]	Alters an existing HTTP over SOAP or DISH service.
COMMENT statement	Stores a comment for a database object in the system tables. Use the following syntax to comment on a web service: <pre>COMMENT ON SERVICE 'web-service-name' IS 'your comments'</pre>
DROP SERVICE statement	Drops a web service.

Connection Pooling for Web Services

Each database that exposes web services has access to a pool of database connections. The pool is grouped by user name such that all services defined under a given USER clause share the same connection pool group.

A service request executing a query for the first time must go through an optimization phase to establish an execution plan. If the plan can be cached and reused, then the optimization phase can be skipped for subsequent executions. HTTP connection pooling leverages the plan caching in database connections by reusing them whenever possible. Each service maintains its own list of connections to optimize reuse. However, during peak loads, a service can steal least utilized connections from within the same user group of connections.

Over time, a given connection may acquire cached plans that can optimize performance for the execution of a number of services.

In a connection pool, an HTTP request for a given service tries to acquire a database connection from a pool. Unlike HTTP sessions, pooled connections are sanitized by resetting the connection scope environment, such as connection scope variables and temporary tables.

Database connections that are pooled within the HTTP connection pool are not counted as connections in use for the purposes of licensing. Connections are counted as licensed connections when they are acquired from the pool. A 503 Service Temporarily Unavailable status is returned when an HTTP request exceeds the licensing restrictions while acquiring a connection from the pool.

Web services can only utilize a connection pool when they are defined with `AUTHORIZATION OFF`.

A database connection within a pool is not updated when changes occur to database and connection options.

How to Develop Web Service Applications in an HTTP Web Server

This section provides an overview of web page creation and customization. It explains how to develop stored procedures for your HTTP web server. It assumes that you have knowledge of starting SAP Sybase IQ HTTP web servers and creating web services that call stored procedures.

For detailed examples of web service applications, see the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP` directory.

How to Customize Web Pages

You must first evaluate the format of the web service invoked by the HTTP web server to customize your web pages. For example, web pages are formatted in HTML when the web service specifies the HTML type.

The RAW web service type provides the most customization because it requires that web service procedures and functions explicitly require coding to provide the required markup such as HTML or XML. The following tasks must be performed to customize web pages when using the RAW type:

- Set the HTTP Content-Type header field to the appropriate MIME type, such as text/html, in the called stored procedure.
- Apply appropriate markup for the MIME type when generating web page output from the called stored procedure.

Example

The following example illustrates how to create a new web service with the RAW type specified:

```
CREATE SERVICE WebServiceName
  TYPE 'RAW'
  AUTHORIZATION OFF
  URL ON
  USER DBA
  AS CALL HomePage( :url );
```

In this example, the web service calls the `HomePage` stored procedure, which is required to define a single URL parameter that receives the PATH component of the URL.

Setting the Content-Type header field

Use the `sa_set_http_header` system procedure to define the HTTP Content-Type header to ensure that web browsers correctly render the content.

The following example illustrates how to format web page output in HTML using the text/html MIME-type with the sa_set_http_header system procedure:

```
CREATE PROCEDURE HomePage (IN url LONG VARCHAR)
  RESULT (html_doc XML)
  BEGIN
    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    -- Your SQL code goes here.
    ...
  END
```

Applying tagging conventions of the MIME-type

You must apply the tagging conventions of the MIME-type specified by the Content-Type header in your stored procedure. SAP Sybase IQ provides several functions that allow you to create tags.

The following example illustrates how to use the XMLCONCAT, and XMLELEMENT functions to generate HTML content, assuming that the sa_set_http_header system procedure is used to set the Content-Type header to the text/html MIME-type:

```
XMLCONCAT (
  CAST ('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' AS XML),
  XMLELEMENT (
    'HTML',
    XMLELEMENT (
      'HEAD',
      XMLELEMENT ('TITLE', 'My Home Page')
    ),
    XMLELEMENT (
      'BODY',
      XMLELEMENT ('H1', 'My home on the web'),
      XMLELEMENT ('P', 'Thank you for visiting my web site!')
    )
  )
)
```

Since element content is always escaped unless the data type is XML, the above example uses the CAST function. Otherwise, special characters are escaped (for example, < for <).

How to Access Client-Supplied HTTP Variables and Headers

Variables and headers in an HTTP client request can be accessed using one of the following approaches:

- The web service statement declaration to pass them as host parameters of a stored function and procedure call.
- Calling the HTTP_VARIABLE, NEXT_HTTP_VARIABLE, HTTP_HEADER, NEXT_HTTP_HEADER functions in a stored function or procedure.

How to Access HTTP Variables Using Host Parameters

You can reference client-supplied variables when you pass them as host parameters of a function or procedure call.

Example

The following example illustrates how to access the host parameters used in a web service named ShowTable:

```
CREATE SERVICE ShowTable
  TYPE 'RAW'
  AUTHORIZATION ON
  AS CALL ShowTable( :user_name, :table_name );

CREATE PROCEDURE ShowTable(IN username VARCHAR(128), IN tblname
  VARCHAR(128))
BEGIN
  -- write SQL code utilizing the username and tblname variables
  here.
END;
```

Service host parameters are mapped in the declaration order of procedure parameters. In the above example, the `user_name` and `table_name` host parameters map to the `username` and `tblname` parameters, respectively.

How to Access HTTP Variables and Headers Using Web Service Functions

The `HTTP_VARIABLE`, `NEXT_HTTP_VARIABLE`, `HTTP_HEADER`, `NEXT_HTTP_HEADER` functions can be used to iterate through the variables and headers supplied by the client.

Accessing variables using HTTP_VARIABLE and HTTP_NEXT_VARIABLE

You can iterate through all client-supplied variables using `NEXT_HTTP_VARIABLE` and `HTTP_VARIABLE` functions within your stored procedures.

The `HTTP_VARIABLE` function allows you to get the value of a variable name.

The `NEXT_HTTP_VARIABLE` function allows you to iterate through all variables sent by the client. Pass the `NULL` value when calling it for the first time to get the first variable name. Use the returned variable name as a parameter to an `HTTP_VARIABLE` function call to get its value. Passing the previous variable name to the `next_http_variable` call gets the next variable name. `Null` is returned when the last variable name is passed.

Iterating through the variable names guarantees that each variable name is returned exactly once but the variable name order may not be the same as the order they appear in the client request.

The following example illustrates how to use the `HTTP_VARIABLE` function to retrieve values from parameters supplied in a client request that accesses the `ShowDetail` service:

```

CREATE SERVICE ShowDetail
  TYPE 'HTML'
  URL PATH OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowDetail();

CREATE PROCEDURE ShowDetail()
BEGIN
  DECLARE v_customer_id LONG VARCHAR;
  DECLARE v_product_id LONG VARCHAR;
  SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
  SET v_product_id = HTTP_VARIABLE( 'product_id' );
  CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;

```

The following example illustrates how to retrieve three attributes from header-field values associated with the `image` variable:

```

SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );

```

Supplying an integer as the second parameter allows you to retrieve additional values. The third parameter allows you to retrieve header-field values from multi-part requests. Supply the name of a header field to retrieve its value.

Accessing headers using HTTP_HEADER and NEXT_HTTP_HEADER

HTTP request headers can be obtained from a request using the `NEXT_HTTP_HEADER` and `HTTP_HEADER` functions.

The `HTTP_HEADER` function returns the value of the named HTTP header field.

The `NEXT_HTTP_HEADER` function iterates through the HTTP headers and returns the next HTTP header name. Calling this function with `NULL` causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the function. `NULL` is returned when the last header name is called.

The following table lists some common HTTP request headers and typical values:

Header name	Header value
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language	en-us
Accept-Charset	utf-8, iso-8859-5;q=0.8
Accept-Encoding	gzip, deflate

Header name	Header value
User-Agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2; WOW64; SV1; .NET CLR 2.0.50727)
Host	localhost:8080
Connection	Keep-Alive

The following table lists special headers and typical values:

Header Name	Header Value
@HttpMethod	GET
@HttpURI	/demo/ShowHTTPHeaders
@HttpVersion	HTTP/1.1
@HttpQueryString	id=-123&version=109&lang=en

You can use the @HttpStatus special header to set the status code of the request being processed.

The following example illustrates how to format header names and values into an HTML table.

Create the ShowHTTPHeaders web service:

```
CREATE SERVICE ShowHTTPHeaders
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL HTTPHeaderExample();
```

Create a HTTPHeaderExample procedure that uses the NEXT_HTTP_HEADER function to get the name of the header, then uses the HTTP_HEADER function to retrieve its value:

```
CREATE PROCEDURE HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
  declare header_name LONG VARCHAR;
  declare header_value LONG VARCHAR;
  declare header_query LONG VARCHAR;
  declare table_rows XML;
  set header_name = NULL;
  set table_rows = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    SET header_query = HTTP_HEADER( '@HttpQueryString' );
```

```

-- Format header name and value into an HTML table row
SET table_rows = table_rows ||
    XMLELEMENT( name "tr",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        XMLELEMENT( name "td", header_name ),
        XMLELEMENT( name "td", header_value ),
        XMLELEMENT( name "td", header_query ) );

END LOOP;
SELECT XMLELEMENT( name "table",
    XMLATTRIBUTES( '' AS "BORDER",
                    '10' AS "CELLPADDING",
                    '0' AS "CELLSPACING" ),
    XMLELEMENT( name "th",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        'Header Name' ),
    XMLELEMENT( name "th",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        'Header Value' ),
    XMLELEMENT( name "th",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        'HTTP Query String' ),
    table_rows );
END;

```

Access the ShowHTTPHeaders in a web browser to see the request headers arranged in an HTML table.

How to Access Client-Supplied SOAP Request Headers

Headers in SOAP requests can be obtained using a combination of the NEXT_SOAP_HEADER and SOAP_HEADER functions.

The NEXT_SOAP_HEADER function iterates through the SOAP headers included within a SOAP request envelope and returns the next SOAP header name. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the NEXT_SOAP_HEADER function. This function returns NULL when called with the name of the last header.

The following example illustrates the SOAP header retrieval:

```

SET hd_key = NEXT_SOAP_HEADER( hd_key );
IF hd_key IS NULL THEN
    -- no more header entries
    LEAVE header_loop;
END IF;

```

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order they appear in the SOAP request.

The SOAP_HEADER function returns the value of the named SOAP header field, or NULL if not called from an SOAP service. It is used when processing an SOAP request via a web service. If a header for the given field-name does not exist, the return value is NULL.

The example searches for a SOAP header named Authentication. When it finds this header, it extracts the value for entire SOAP header and the values of the @namespace and mustUnderstand attributes. The SOAP header value might look something like this XML string:

```
<Authentication xmlns="CustomerOrderURN" mustUnderstand="1">  
  <userName pwd="none">  
    <first>John</first>  
    <last>Smith</last>  
  </userName>  
</Authentication>
```

For this header, the @namespace attribute value would be CustomerOrderURN

Also, the mustUnderstand attribute value would be 1

The interior of this XML string is parsed with the OPENXML function using an XPath string set to /*:Authentication/*:userName.

```
SELECT * FROM OPENXML( hd_entry, xpath )  
  WITH ( pwd LONG VARCHAR '@*:pwd',  
         first_name LONG VARCHAR '*:first/text()',  
         last_name LONG VARCHAR '*:last/text()' );
```

Using the sample SOAP header value shown above, the SELECT statement would create a result set as follows:

pwd	first_name	last_name
none	John	Smith

A cursor is declared on this result set and the three column values are fetched into three variables. At this point, you have all the information of interest that was passed to the web service.

Example

The following example illustrates how a web server can process SOAP requests containing parameters, and SOAP headers. The example implements an addItem SOAP operation that takes two parameters: amount of type int and item of type string. The sp_addItems procedure processes an Authentication SOAP header extracting the first and last name of the user. The values are used to populate a SOAP response Validation header via the sa_set_soap_header system procedure. The response is a result of three columns: quantity, item and status with types INT, LONG VARCHAR and LONG VARCHAR respectively.

```
// create the SOAP service  
CREATE SERVICE addItem  
  TYPE 'SOAP'  
  FORMAT 'CONCRETE'  
  AUTHORIZATION OFF
```

```

USER DBA
AS CALL sp_addItems( :amount, :item );

// create SOAP endpoint for related services
CREATE SERVICE itemStore
  TYPE 'DISH'
  AUTHORIZATION OFF
  USER DBA;

// create the procedure that will process the SOAP requests for the
addItems service
CREATE PROCEDURE sp_addItems(count INT, item LONG VARCHAR)
RESULT(quantity INT, item LONG VARCHAR, status LONG VARCHAR)
BEGIN
  DECLARE hd_key LONG VARCHAR;
  DECLARE hd_entry LONG VARCHAR;
  DECLARE pwd LONG VARCHAR;
  DECLARE first_name LONG VARCHAR;
  DECLARE last_name LONG VARCHAR;
  DECLARE xpath LONG VARCHAR;
  DECLARE authinfo LONG VARCHAR;
  DECLARE namespace LONG VARCHAR;
  DECLARE mustUnderstand LONG VARCHAR;

  header_loop:
  LOOP
    SET hd_key = next_soap_header( hd_key );
    IF hd_key IS NULL THEN
      // no more header entries.
      leave header_loop;
    END IF;
    IF hd_key = 'Authentication' THEN
      SET hd_entry = soap_header( hd_key );
      SET xpath = '/*:' || hd_key || '/*:userName';
      SET namespace = soap_header( hd_key, 1, '@namespace' );
      SET mustUnderstand = soap_header( hd_key, 1,
'mustUnderstand' );
      BEGIN
        // parse for the pieces that you are interested in
        DECLARE crsr CURSOR FOR SELECT * FROM
          OPENXML( hd_entry, xpath )
            WITH ( pwd LONG VARCHAR '@:pwd',
                  first_name LONG VARCHAR '*:first/text()',
                  last_name LONG VARCHAR '*:last/text()' );
        OPEN crsr;
        FETCH crsr INTO pwd, first_name, last_name;
        CLOSE crsr;
      END;
      // build a response header, based on the pieces from the
request header
      SET authinfo = XMLELEMENT( 'Validation',
        XMLATTRIBUTES (
          namespace as xmlns,
          mustUnderstand as mustUnderstand ),
        XMLELEMENT( 'first', first_name ),
        XMLELEMENT( 'last', last_name ) );

```

```
        CALL sa_set_soap_header( 'authinfo', authinfo);
    END IF;
END LOOP header_loop;
// code to validate user/session and check item goes here...
SELECT count, item, 'available';
END;
```

HTTP Session Management on an HTTP Server

A web application can support sessions in various ways. Hidden fields within HTML forms can be used to preserve client/server data across multiple requests. Alternatively, Web 2.0 techniques, such as an AJAX enabled client-side JavaScript, can make asynchronous HTTP requests based on client state. SAP Sybase IQ offers the additional capability of preserving a database connection for exclusive use of sessioned HTTP requests.

Any connection scope variables and temporary tables created and altered within the HTTP session are accessible to subsequent HTTP requests that specify the given SessionID. The SessionID can be specified by a GET or POST HTTP request method or specified within an HTTP cookie header. When an HTTP request is received with a SessionID variable, the server checks its session repository for a matching context. If it finds a session, the server utilizes its database connection for processing the request. If the session is in use, it queues the HTTP request and activates it when the session is freed.

The `sa_set_http_option` can be used to create, delete and change session ids.

HTTP sessions require special handling for management of the session criteria. Only one database connection exists for use by a given SessionID, so consecutive client requests for that SessionID are serialized by the server. Up to 16 requests can be queued for a given SessionID. Subsequent requests for the given SessionID are rejected with a 503 Service Unavailable status when the session queue is full.

When creating a SessionID for the first time, the SessionID is immediately registered by the system. Subsequent requests that modify or delete the SessionID are only applied when the given HTTP request terminates. This approach promotes consistent behavior if the request processing results in a roll-back or if the application deletes and resets the SessionID.

The current session is deleted and replaced with the pending session when an HTTP request changes the SessionID. The database connection cached by the session is effectively moved to the new session context, and all state data is preserved, such as temporary tables and created variables.

For a complete example of HTTP session usage, see `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\session.sql`.

Note: Stale sessions should be deleted and an appropriate timeout should be set to minimize the number of outstanding connections because each client application connection holds a license seat. Connections associated with HTTP sessions maintain their hold on the server database for the duration of the connection.

For more information about licensing, see <http://www.sybase.com/detail?id=1056242>.

How to Create an HTTP Session

Sessions can be created using the `SessionID` option in the `sa_set_http_option` system procedure. The session ID can be defined by any non-null string.

Session state management is supported by URLs and cookies. HTTP sessions can be accessed using HTTP cookies, or through the URL of a GET request or from within the body of a POST (x-www-form-urlencoded) request. For example, the following URL utilizes the `XYZ` database connection when it executes:

```
http://localhost/sa_svc?SESSIONID=XYZ
```

The request is processed as a standard session-less request if an `XYZ` database connection does not exist.

Example

The following code illustrates how to create a RAW web service that creates and deletes sessions. A connection scope variable named `request_count` is incremented each time an HTTP request is made while specifying a valid `SessionID`.

```
CREATE SERVICE mysession
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL mysession_proc();

CREATE PROCEDURE mysession_proc()
BEGIN
  DECLARE body LONG VARCHAR;
  DECLARE hostname LONG VARCHAR;
  DECLARE svcname LONG VARCHAR;
  DECLARE sesid LONG VARCHAR;

  CALL sa_set_http_header ( 'Content-Type', 'text/html' );
  SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
  SELECT CONNECTION_PROPERTY('HttpServiceName') INTO svcname;
  SELECT HTTP_HEADER( 'Host' ) INTO hostname;
  IF HTTP_VARIABLE('delete') IS NOT NULL THEN
    CALL sa_set_http_option( 'SessionID', NULL );
    SET body = '<html><body>Deleted ' || sesid
      || '</BR><a href="http://' || hostname || '/' || svcname ||
'">Start Again</a>';
    SELECT body;
  END IF;
  IF sesid = '' THEN
    SET sesid = set_session_url();
    CREATE VARIABLE request_count INT;
    SET request_count = 0;

    SET body = '<html><body> Created session ID ' || sesid
      || '</br><a href="http://' || hostname || '/' || svcname
      || '?SessionID=' || sesid || '"> Enter into Session</a>';
  ELSE

```

```

        SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
        SET request_count = request_count + 1;
        SET body = '<html><body>Session ' || sesid || '</br>'
        || 'created ' || CONNECTION_PROPERTY('SessionCreateTime')
        || '</br>'
        || 'last access ' ||
CONNECTION_PROPERTY('SessionLastTime') || '</br>'
        || 'connection ID ' || CONNECTION_PROPERTY('Number') ||
'</br>'
        || '<h3>REQUEST COUNT is ' || request_count || '</h3><hr></
br>'
        || '<a href="http://' || hostname || '/' || svcname
        || '?SessionID=' || sesid || '">Enter into Session</a></
br>'
        || '<a href="http://' || hostname || '/' || svcname
        || '?SessionID=' || sesid || '&delete">Delete Session</
a>';
        END IF;

        SELECT body;
    END;

```

How to Use the URL to Manage a Session

In a URL session state management system, the client application or web browser provides the session ID in a URL.

Example

The following example illustrates unique session ID creation within an HTTP web server SQL function where session IDs can be provided by a URL only:

```

CREATE FUNCTION set_session_url()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
        CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND,
tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;

```

The SessionID is represented as an empty string if the session_id is not defined for the connection, making a sessionless connection.

The sa_set_http_option system procedure returns an error if the session_id is owned by another HTTP request.

How to Use Cookies to Manage a Session

In a cookie session state management system, the client application or web browser provides the session ID in an HTTP cookie header instead of a URL. Cookie session management is

supported with the 'Set-Cookie' HTTP response header of the `sa_set_http_header` system procedure.

Note: You cannot rely on cookie state management when cookies can be disabled in the client application or web browser. Support for both URL and cookie state management is recommended. The URL-supplied session ID is used when session IDs are provided by both the URL and a cookie.

Example

The following example illustrates unique session ID creation within an HTTP web server SQL function where session IDs can be provided by a URL or a cookie:

```
CREATE FUNCTION set_session_cookie()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
        CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND,
tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    CALL sa_set_http_header( 'Set-Cookie',
        'sessionid=' || session_id || ';' ||
        'max-age=60;' ||
        'path=/session;' );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

How to Detect an Inactive HTTP Session

The `SessionCreateTime` and `SessionLastTime` connection properties can be used to determine if the current connection is within a session context. The HTTP request is not running within a session context when either connection property query returns an empty string.

The `SessionCreateTime` connection property provides a metric of when a given session was created. It is initially defined when the `sa_set_http_option` system procedure is called to establish the `SessionID`.

The `SessionLastTime` connection property provides the time when the last processed session request released the database connection upon termination of the previous request. It is returned as an empty string when the session is first created until the creator request releases the connection.

Note: You can adjust the session timeout duration using the `http_session_timeout` option.

Example

The following example illustrates session detection using the `SessionCreateTime` and `SessionLastTime` connection properties:

```
SELECT CONNECTION_PROPERTY( 'sessioncreatetime' ) INTO ses_create;  
SELECT CONNECTION_PROPERTY( 'sessionlasttime' ) INTO ses_last;
```

How to Delete an HTTP Session or Change the Session ID

Explicitly dropping a database connection that is cached within a session context causes the session to be deleted. Session deletion in this manner is a cancel operation; any requests released from the session queue are in a canceled state. This action ensures that any outstanding requests waiting on the session are terminated. Similarly, a server or database shutdown cancels all database connections.

A session can be deleted by setting the SessionID option in the sa_set_http_option system procedure to null or an empty string.

The following code can be used for session deletion:

```
CALL sa_set_http_option( 'SessionID', null );
```

When an HTTP session is deleted or the SessionID is changed, any pending HTTP requests that are waiting on the session queue are released and allowed to run outside of a session context. The pending requests do not reuse the same database connection.

A session ID cannot be set to an existing session ID. Pending requests referring to the old SessionID are released to run as session-less requests when a SessionID has changed. Subsequent requests referring to the new SessionID reuse the same database connection instantiated by the old SessionID.

The following conditions are applied when deleting or changing an HTTP session:

- The behavior differs depending on whether the current request had inherited a session whereby a database connection belonging to a session was acquired, or whether a session-less request had instantiated a new session. If the request began as session-less, then the act of creating or deleting a session occurs immediately. If the request has inherited a session, then a change in the session state, such as deleting the session or changing the SessionID, only occurs after the request terminates and its changes have been committed. The difference in behavior addresses processing anomalies that may occur if a client makes simultaneous requests using the same SessionID.
- Changing a session to a SessionID of the current session (has no pending session) is not an error and has no substantial effect.
- Changing a session to a SessionID in use by another HTTP request is an error.
- Changing a session when a change is already pending results in the pending session being deleted and new pending session being created. The pending session is only activated once the request successfully terminates.
- Changing a session with a pending session back to its original SessionID results in the pending session being deleted without any change to the current session.

HTTP Session Administration

A session created by an HTTP request is immediately instantiated so that any subsequent HTTP requests requiring that session context is queued by the session.

In this example, a local host client can access the session with the specified session ID, session_63315422814117, running within the database, dbname, running the service session_service with the following URL once the session is created on the server with the sa_set_http_option procedure.

```
http://localhost/dbname/session_service?
sessionid=session_63315422814117
```

A web application can require a means to track active session usage within the HTTP web server. Session data can be found using the NEXT_CONNECTION function call to iterate through the active database connections and checking for session related properties such as SessionID.

The following SQL statements illustrate how to track an active session:

```
CREATE VARIABLE conn_id LONG VARCHAR;
CREATE VARIABLE the_sessionID LONG VARCHAR;
SELECT NEXT_CONNECTION( NULL, NULL ) INTO conn_id;
conn_loop:
  LOOP
    IF conn_id IS NULL THEN
      LEAVE conn_loop;
    END IF;
    SELECT CONNECTION_PROPERTY( 'SessionID', conn_id )
      INTO the_sessionID;
    IF the_sessionID != '' THEN
      PRINT 'conn_id = %1!, SessionID = %2!', conn_id,
the_sessionID;
    ELSE
      PRINT 'conn_id = %1!', conn_id;
    END IF;
    SELECT NEXT_CONNECTION( conn_id, NULL ) INTO conn_id;
  END LOOP conn_loop;
PRINT '\n';
```

If you examine the database server messages window, you see data that is similar to the following output:

```
conn_id = 30
conn_id = 29, SessionID = session_63315442223323
conn_id = 28, SessionID = session_63315442220088
conn_id = 25, SessionID = session_63315441867629
```

Explicitly dropping a connection that belongs to a session causes the connection to be closed and the session to be deleted. If the connection being dropped is currently active in servicing an HTTP request, the request is marked for deletion and the connection is sent a cancel signal to terminate the request. When the request terminates, the session is deleted and the

connection closed. Deleting the session causes any pending requests on that session's queue to be re-queued.

In the event the connection is currently inactive, the session is marked for deletion and re-queued to the beginning of the session timeout queue. The session and the connection are deleted in the next timeout cycle (normally within 5 seconds). Any session marked for deletion cannot be used by a new HTTP request.

All sessions are lost when the database is stopped.

HTTP Session Error Codes

The 503 `Service Unavailable` error occurs when a new request tries to access a session where more than 16 requests are pending on that session, or an error occurred while queuing the session.

The 403 `Forbidden` error occurs when the client IP address or host name does not match that of the creator of the session.

A request stipulating a session that does not exist does not implicitly generate an error. It is up to the web application to detect this condition (by checking `SessionID`, `SessionCreateTime`, or `SessionLastTime` connection properties) and do the appropriate action.

Character Set Conversion Considerations

Character-set conversion is performed automatically on outgoing result sets of text types by default. Result sets of other types, such as binary objects, are not affected. The character set of the request is converted to the HTTP web server character set, and the result set is converted to the client application character set. The server uses the first suitable character set listed in the request when multiple sets are listed.

Character-set conversion can be enabled or disabled by setting the HTTP option 'CharsetConversion' option of the `sa_set_http_option` system procedure.

The following example illustrates how to turn off automatic character-set conversion:

```
CALL sa_set_http_option('CharsetConversion', 'OFF');
```

You can use the 'AcceptCharset' option of the `sa_set_http_option` system procedure to specify the character-set encoding preference when character-set conversion is enabled.

The following example illustrates how to specify the web service character set encoding preference to ISO-8859-5, if supported; otherwise, set it to UTF-8:

```
CALL sa_set_http_option('AcceptCharset', 'iso-8859-5, utf-8');
```

Character sets are prioritized by server preference but the selection also considers the client's Accept-Charset criteria. The most favored character set according to the client that is also specified by this option is used.

Cross Site Scripting Considerations

When developing your web application, you should ensure that it is not vulnerable to cross-site scripting (XSS). This type of vulnerability occurs when an attacker attempts to inject a script into your web page.

It is highly recommended that application developers and database administrators review their web application code for possible security vulnerabilities before it is put into production. The Open Web Application Security Project (<https://www.owasp.org>) contains more information about how to secure your web application.

Web Services System Procedures

The following system procedures are for use with web services:

- sa_http_header_info system procedure
- sa_http_php_page system procedure
- sa_http_php_page_interpreted system procedure
- sa_http_variable_info system procedure
- sa_set_http_header system procedure
- sa_set_http_option system procedure
- sa_set_soap_header system procedure

Web Services Functions

Web service functions assist the handling of HTTP and SOAP requests within web services.

The following functions are available:

- HTML_DECODE function [Miscellaneous]
- HTML_ENCODE function [Miscellaneous]
- HTTP_BODY function [Web service]
- HTTP_DECODE function [Web service]
- HTTP_ENCODE function [Web service]
- HTTP_HEADER function [Web service]
- HTTP_RESPONSE_HEADER function [Web service]
- HTTP_VARIABLE function [Web service]
- NEXT_HTTP_HEADER function [Web service]
- NEXT_HTTP_RESPONSE_HEADER function [Web service]
- NEXT_HTTP_VARIABLE function [Web service]
- NEXT_SOAP_HEADER function [SOAP]
- SOAP_HEADER function [SOAP]

There are also many system procedures available for web services.

Web Services Connection Properties

Web service connection properties can be database properties that are accessible using the CONNECTION_PROPERTY function.

Use the following syntax to store a connection property value from the HTTP server to a local variable in a SQL function or procedure:

```
SELECT CONNECTION_PROPERTY('connection-property-name') INTO  
variable_name;
```

The following is a list of useful runtime HTTP request connection properties that are commonly used for web service applications:

- **HttpServiceName** – Returns the service name origin for a web application.
- **AuthType** – Returns the type of authentication used when connecting.
- **ServerPort** – Returns the database server's TCP/IP port number or 0.
- **ClientNodeAddress** – Returns the node for the client in a client/server connection.
- **ServerNodeAddress** – Returns the node for the server in a client/server connection.
- **BytesReceived** – Returns the number of bytes received during client/server communications.

Web Services Options

Web service options control various aspects of HTTP server behavior.

Use the following syntax to set a public option in an HTTP server:

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

The following is a list of options that are commonly used in HTTP servers for application configuration:

- **http_connection_pool_basesize** – Specifies the nominal threshold size of database connections.
- **http_connection_pool_timeout** – Specifies the maximum duration that an unused connection can be retained in the connection pool.
- **http_session_timeout** – Specifies the default timeout duration, in minutes, that the HTTP session persists during inactivity.
- **request_timeout** – Controls the maximum time a single request can run.
- **webservice_namespace_host** – Specifies the hostname to be used as the XML namespace within specification for DISH services.

How to Browse the SAP Sybase IQ HTTP Web Server

Available URL names are defined by how your web services are named and designed. Each web service provides its own set of web content. This content is typically generated by custom

functions and procedures in your database, but content can also be generated with a URL that specifies a SQL statement.

Alternatively, or in conjunction, you can define the `root` web service, which processes all HTTP requests that are not processed by a dedicated service. The `root` web service would typically inspect the request URL and headers to determine how to process the request.

URLs uniquely specify resources such as html content available through HTTP or secured HTTPS requests. This section explains how to format the URL syntax in your web browser so that you can access the web services defined on your SAP Sybase IQ HTTP web server.

Note: The information in this section applies to HTTP web servers that use general HTTP web service types, such as RAW, XML, and HTML, and DISH services. You cannot use a browser to issue SOAP requests. JSON services return result sets for consumption by web service applications using AJAX.

Syntax

```
{http|https}://host-name[:port-number][/dbn]/service-name[/path-name|?url-query]
```

Parameters

- **host-name and port-number** – Specifies the location of the web server and, optionally, the port number if it is not defined as the default HTTP or HTTPS port numbers. The *host-name* can be the IP address of the computer running the web server. The *port-number* must match the port number used when you started the web server.
- **dbn** – Specifies the name of a database. This database must be running on the web server and contain web services.

You do not need to specify *dbn* if the web server is running only one database or if the database name was specified for the given HTTP/HTTPS listener of the protocol option.

- **service-name** – Specifies the name of the web service to access. This web service must exist in the database specified by *dbn*. Slash characters (/) are permitted when you create or alter a web service, so you can use them as part of the *service-name*. SAP Sybase IQ matches the remainder of the URL with the defined services.

The client request is processed if a *service-name* is not specified and the `root` web service is defined. A 404 Not Found error is returned if the server cannot identify an applicable service to process the request. As a side-effect, if the `root` web service does exist and cannot process the request based on the URL criteria, then it is responsible for generating the 404 Not Found error.

- **path-name** – After resolving the service name, the remaining slash delimited path can be accessed by a web service procedure. If the service was created with URL ON, then the whole path is accessible using a designated URL HTTP variable. If the service was created with URL ELEMENTS, then each path element can be accessed using designated HTTP variables URL1 to URL10.

Path element variables can be defined as host variables within the parameter declaration of the service statement definition. Alternatively, or additionally, HTTP variables can be accessed from within a stored procedure using the `HTTP_VARIABLE` function call.

The following example illustrates the SQL statement used to create a web service where the `URL` clause is set to `ELEMENTS`:

```
CREATE SERVICE TestWebService
  TYPE 'HTML'
  URL ELEMENTS
  AUTHORIZATION OFF
  USER DBA
  AS CALL TestProcedure ( :url1, :url2 );
```

This `TestWebService` web service calls a procedure that explicitly references the `url1` and `url2` host variables.

You can access this web service using the following URL, assuming that `TestWebService` is running on the demo database from `localhost` through the default port:

```
http://localhost/demo/TestWebService/Assignment1/Assignment2/Assignment3
```

This URL accesses `TestWebService`, which runs `TestProcedure` and assigns the `Assignment1` value to `url1`, and the `Assignment2` value to `url2`. Optionally, `TestProcedure` can access other path elements using the `HTTP_VARIABLE` function. For example, the `HTTP_VARIABLE('url3')` function call returns `Assignment3`.

- **url-query** – An HTTP GET request may follow a path with a query component that specifies HTTP variables. Similarly, the body of a POST request using a standard `application/x-www-form-urlencoded` Content-Type can pass HTTP variables within the request body. In either case, HTTP variables are passed as name/value pairs where the variable name is delimited from its value with an equals sign. Variables are delimited with an ampersand.

HTTP variables can be explicitly declared as host variables within the parameter list of the service-statement, or accessed using the `HTTP_VARIABLE` function from within the stored procedure of the service statement.

For example, the following SQL statement creates a web service that requires two host variables. Host variables are identified with a colon (`:`) prefix.

```
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

Assuming that `ShowSalesOrderDetail` is running on the demonstration database from `localhost` through the default port, you can access the web service using the following URL:

```
http://localhost/demo/ShowSalesOrderDetail?
customer_id=101&product_id=300
```

This URL accesses `ShowSalesOrderDetail` and assigns a value of 101 to `customer_id`, and a value of 300 to `product_id`. The resultant output is displayed in your web browser in HTML format.

Remarks

The web browser prompts for user name and password when required to connect to the server. The browser then base64 encodes the user input within an Authorization request header and resends the request.

If your web service URL clause is set to ON or ELEMENTS, the URL syntax properties of *path-name* and *url-query* can be used simultaneously so that the web service is accessible using one of several different formatting options. When using these syntax properties simultaneously, the *path-name* format must be used first followed by the *url-query* format.

In the following example, this SQL statement creates a web service where the URL clause is set to ON, which defines the `url` variable:

```
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :product_id, :url );
```

The following is a sample list of acceptable URLs that assign a `url` value of 101 and a `product_id` value of 300:

```
http://localhost:80/demo/ShowSalesOrderDetail2/101?
product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2?
url=101&product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2?
product_id=300&url=101
```

When a host variable name is assigned more than once in the context of *path-name* and *url-query*, the last assignment always takes precedence. For example, the following sample URLs assign a `url` value of 101 and a `product_id` value of 300:

```
http://localhost:80/demo/ShowSalesOrderDetail2/302?
url=101&product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2/String?
product_id=300&url=101
```

Example

The following URL syntax is used to access a web service named `gallery_image` that is running in a database named `demo` on a local HTTP server through the default port, assuming that the `gallery_image` service is defined with URL ON:

```
http://localhost/demo/gallery_image/sunset.jpg
```

The URL appears to request a graphic file in a directory from a traditional web server, but it accesses the `gallery_image` service with `sunset.jpg` specified as an input parameter for an HTTP web server.

The following SQL statement illustrates how the `gallery` service could be defined on the HTTP server to accomplish this behavior:

```
CREATE SERVICE gallery_image
  TYPE 'RAW'
  URL ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL gallery_image ( :url );
```

The `gallery_image` service calls a procedure with the same name, passing the client-supplied URL. For a sample implementation of a `gallery_image` procedure that can be accessed by this web service definitions, see `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\gallery.sql`.

Access to Web Services Using Web Clients

SAP Sybase IQ can be used as a web client to access web services hosted by an SAP Sybase IQ web server or third party web servers such as Apache or IIS.

In addition to using SAP Sybase IQ as a web client, SAP Sybase IQ web services provide client applications with an alternative to traditional interfaces, such as JDBC and ODBC. They are easily deployed because additional components are not needed, and can be accessed from multi-platform client applications written in a variety of languages, including scripting languages — such as Perl and Python.

Quick Start to Using SAP Sybase IQ As a Web Client

This section illustrates how to use SAP Sybase IQ as a web client application to connect to an SAP Sybase IQ HTTP server and access a general HTTP web service. It does not illustrate SAP Sybase IQ web client capabilities to a full extent. Many SAP Sybase IQ web client features are available that are beyond the scope of this topic.

You can develop SAP Sybase IQ web client applications that connect to any type of online web server, but this section assumes that you have started a local SAP Sybase IQ HTTP server on port 8082 and want to connect to a web service named `SampleHTMLService`, created with the following SQL statements:

```

CREATE SERVICE SampleHTMLService
  TYPE 'HTML'
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT(ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR)
BEGIN
  SELECT i, f, s;
END;

```

Perform the following tasks to create an SAP Sybase IQ web client application:

1. Run the following command to create an SAP Sybase IQ client database if one does not already exist:

```
iqinit -dba DBA,sql client-database-name
```

Replace *client-database-name* with a new name for your client database.

2. Run the following command to start the client database:

```
iqsrvl6 client-database-name.db
```

3. Run the following command to connect to the client database through Interactive SQL:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=client-database-name"
```

4. Create a new client procedure that connects to the SampleHTMLService web service using the following SQL statement:

```

CREATE PROCEDURE client_post(f REAL, i INTEGER, s VARCHAR(16), x
VARCHAR(16))
  URL 'http://localhost:8082/SampleHTMLService'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SATest';

```

5. Execute the following SQL statement to call the client procedure and send an HTTP request to the web server:

```
CALL client_post(3.14, 9, 's varchar', 'x varchar');
```

The HTTP POST request created by `client_post` looks similar to the following output:

```

POST /SampleHTMLService HTTP/1.0
ASA-Id: eal746b01cd0472eb4f0729948db60a2
User-Agent: SATest
Accept-Charset: windows-1252, UTF-8, *
Date: Wed, 9 Jun 2010 21:55:01 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 58

&f=3.1400001049041748&i=9&s=s%20varchar&x=x%20varchar

```

The web service `SampleHTMLService` running on the web server extracts the parameter values for `i`, `f`, and `s` from the POST request and passes them as parameters to the `sp_echo` procedure. Parameter value `x` is ignored. The `sp_echo` procedure creates a result set which is

returned to the web service. Agreement in parameter names between the client and the web server is essential for proper mapping.

The web service creates the response which is sent back to the client. The output displayed in Interactive SQL should be similar to the following output:

Attribute	Value
Status	HTTP/1.1 200 OK
Body	<pre><html> <head> <title>/SampleHTMLService</ti- title></head> <body> <h3>/SampleHTMLService</h3> <table border=1> <tr class="head- er"><th>ret_i</th> <th>ret_f</th> <th>ret_s</th> </tr> <tr><td>9</ td><td>3.1400001049041748</ td><td>s varchar</td></pre>
Date	Wed, 09 Jun 2010 21:55:01 GMT
Connection	close
Expires	Wed, 09 Jun 2010 21:55:01 GMT
Content-Type	text/html; charset=windows-1252
Server	Sybase IQ/16.0.0

Quick Start to Accessing an SAP Sybase IQ HTTP Web Server

This section illustrates how to access an SAP Sybase IQ HTTP web server using two different types of client application — Python and C#. It does not illustrate SAP Sybase IQ web service application capabilities to a full extent. Many SAP Sybase IQ web service features are available that are beyond the scope of this topic.

You can develop SAP Sybase IQ web client applications that connect to any type of online web server, but this guide assumes that you have started a local SAP Sybase IQ HTTP server on port 8082 and want to connect to a web service named `SampleXMLService`, created with the following SQL statements:

```
CREATE SERVICE SampleXMLService
  TYPE 'XML'
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo2(:i, :f, :s);
```

```
CREATE PROCEDURE sp_echo2(i INTEGER, f NUMERIC(6,2), s LONG VARCHAR )
RESULT( ret_i INTEGER, ret_f NUMERIC(6,2), ret_s LONG VARCHAR )
BEGIN
    SELECT i, f, s;
END;
```

Perform the following tasks to access an XML web service using C# or Python:

1. Create a procedure that connects to a web service on an HTTP server.

Write code that accesses the SampleXMLService web service.

- For C#, use the following code:

```
using System;
using System.Xml;

public class WebClient
{
    static void Main(string[] args)
    {
        XmlTextReader reader = new XmlTextReader(
            "http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello");
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element:
                    if (reader.Name == "row")
                    {
                        Console.Write(reader.GetAttribute("ret_i")
+ " ");
                        Console.Write(reader.GetAttribute("ret_s")
+ " ");
                        Console.WriteLine(reader.GetAttribute("ret_f"));
                    }
                    break;
            }
            reader.Close();
        }
    }
}
```

Save the code to a file named DocHandler.cs.

To compile the program, run the following command at a command prompt:

```
csc /out:DocHandler.exe DocHandler.cs
```

- For Python, use the following code:

```
import xml.sax

class DocHandler( xml.sax.ContentHandler ):
    def startElement( self, name, attrs ):
        if name == 'row':
            table_int = attrs.getValue( 'ret_i' )
            table_string = attrs.getValue( 'ret_s' )
            table_numeric = attrs.getValue( 'ret_f' )
```

```
print('%s %s %s' % ( table_int, table_string,
table_numeric ))

parser = xml.sax.make_parser()
parser.setContentHandler( DocHandler() )
parser.parse('http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello')
```

Save the code to a file named `DocHandler.py`.

2. Perform operations on the result set sent by the HTTP server.

- For C#, run the following command:

```
DocHandler
```

- For Python, run the following command:

```
python DocHandler.py
```

The application displays the following output:

```
5 hello 3.14
```

Web Client Application Development

SAP Sybase IQ databases can act as web client applications to access SAP Sybase IQ hosted web services or web services hosted on third party web servers. SAP Sybase IQ web client applications are created by writing stored procedures and functions using configuration clauses, such as the URL clause that specifies the web service target endpoint. Web client procedures do not have a body, but in every other way are used as any other stored procedure. When called, a web client procedure makes an outbound HTTP or SOAP request. A web client procedure is restricted from making an outbound HTTP request to itself; it cannot call a localhost SAP Sybase IQ web service running on the same database.

For detailed examples of web service applications, see the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP` directory.

Web Client Function and Procedure Requirements and Recommendations

Web service client procedures and functions require the definition of a URL clause to identify the web service endpoint. A web service client procedure or function has specialized clauses for configuration but is used like any other stored procedure or function in every other respect.

You can use the `CREATE PROCEDURE` and `CREATE FUNCTION` statements to create web client functions and procedures to send SOAP or HTTP requests to a web server.

The following list outlines the requirements and recommendations for creating or altering web client functions and procedures. You can specify the following information when creating or altering a web client function or procedure:

- The URL clause, which requires an absolute URL specifying the web service endpoint. (Required)
- The TYPE clause to specify whether the request is HTTP or SOAP over HTTP. (Recommended)

- Ports that are accessible to the client application. (Optional)
- The HEADER clause to specify HTTP request headers. (Optional)
- The SOAPHEADER clause to specify SOAP header criteria within the SOAP request envelope. (Optional. For SOAP requests only)
- The namespace URI. (For SOAP requests only)

Web Client URL Clause

You must specify the location of the web service endpoint to make it accessible to your web client function or procedure. The URL clause of the CREATE PROCEDURE and CREATE FUNCTION statements provides the web service URL that you want to access.

Specifying an HTTP service URL

Specifying an HTTP scheme within the URL clause configures the procedure or function for non-secure communication using an HTTP protocol.

The following statement illustrates how to create a procedure that sends requests to a web service named `SampleHTMLService` that resides in a database named `dbname` hosted by an HTTP web server located at `localhost` on port 8082:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'http://localhost:8082/dbname/SampleHTMLService'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

The database name is only required if the HTTP server hosts more than one database. You can substitute `localhost` with the host name or the IP address of the HTTP server.

Specifying an HTTPS service URL

Specifying an HTTPS scheme within the URL clause configures the procedure or function for secure communication over Secure Socket Layer (SSL).

Your web client application must have access to an RSA server certificate or the certificate that signed the server certificate to issue a secure HTTPS request. The certificate is required for the client procedure to authenticate the server to prevent man-in-the-middle exploits.

Use the CERTIFICATE clause of the CREATE PROCEDURE and CREATE FUNCTION statements to authenticate the server and establish a secure data channel. You can either place the certificate in a file and provide the file name, or provide the entire certificate as a string value; you cannot do both.

The following statement demonstrates how to create a procedure that sends requests to a web service named `SecureHTMLService` that resides in a database named `dbname` in an HTTPS server located at `localhost` on the port 8082:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'HTTPS://localhost:8082/dbname/SecureHTMLService'
    CERTIFICATE 'file=%ALLUSERSPROFILE%/SybaseIQ/demo\Certificates\
\raroot.crt'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

The CERTIFICATE clause in this example indicates that the RSA server certificate is located in the %ALLUSERSPROFILE%\SybaseIQ\samples\Certificates\raroot.crt file.

Note: Specifying HTTPS_FIPS forces the system to use the FIPS libraries. If HTTPS_FIPS is specified, but no FIPS libraries are present, non-FIPS libraries are used instead.

Specifying a proxy server URL

Some requests need to be sent through a proxy server. Use the PROXY clause of the CREATE PROCEDURE and CREATE FUNCTION statements to specify the proxy server URL and redirect requests to that URL. The proxy server forwards the request to the final destination, obtains the response, and forwards the response back to SAP Sybase IQ.

Web Service Request Types

You can specify the type of client requests to send to the web server when creating a web client function or procedure. The TYPE clause of the CREATE PROCEDURE and CREATE FUNCTION statements formats requests before sending them to the web server.

Specifying an HTTP request format

Web client functions and procedures send HTTP requests when the specified format in the TYPE clause begins with an HTTP prefix.

For example, execute the following SQL statement in the web client database to create an HTTP procedure named PostOperation that sends HTTP requests to the specified URL:

```
CREATE PROCEDURE PostOperation(a INTEGER, b CHAR(128))
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  TYPE 'HTTP:POST';
```

In this example, requests are formatted as HTTP:POST requests, which would produce a request similar to the following:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:02:49 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 12

a=123&b=data
```

Specifying a SOAP request format

Web client functions and procedures send HTTP requests when the specified format in the TYPE clause begins with a SOAP prefix.

For example, execute the following statement in the web client database to create a SOAP procedure named `SoapOperation` that sends SOAP requests to the specified URL:

```
CREATE PROCEDURE SoapOperation(intVariable INTEGER, charVariable
CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleSoapService'
    TYPE 'SOAP:DOC';
```

In this example, a `SOAP:DOC` request is sent to the URL when you call this procedure, which would produce a request similar to the following:

```
POST /dbname/SampleSoapService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:05:13 GMT
Host: localhost:8082
Connection: close
Content-Type: text/xml; charset=windows-1252
Content-Length: 428
SOAPAction: "HTTP://localhost:8082/SoapOperation"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The procedure name appears in the `<m:SoapOperation>` tag within the body. The two parameters to the procedure, `intVariable` and `charVariable`, become `<m:intVariable>` and `<m:charVariable>`, respectively.

By default, the stored procedure name is used as the SOAP operation name when building a SOAP request. Parameter names appear in SOAP envelope tag names. You must reference these names correctly when defining a SOAP stored procedure since the server expects these names in the SOAP request. The `SET` clause can be used to specify an alternate SOAP operation name for the given procedure. `WSDL` can be used to read a `WSDL` from a file or URL specification and generate SQL stub functions or procedures. For all but the simplest cases (for example, a SOAP RPC call returning a single string value), it is recommended that function definitions be used rather than procedures. A SOAP function returns the full SOAP response envelope which can be parsed using `OPENXML`.

Web Client Ports

It is sometimes necessary to indicate which ports to use when opening a server connection through a firewall. You can use the `CLIENTPORT` clause of the `CREATE PROCEDURE` and `CREATE FUNCTION` statements to designate port numbers on which the client application communicates using TCP/IP. It is recommended that you not use this feature unless your firewall restricts access to a particular range of ports.

For example, execute the following SQL statement in the web client database to create a procedure named `SomeOperation` that sends requests to the specified URL using one of the ports in the range 5050-5060, or port 5070:

```
CREATE PROCEDURE SomeOperation()  
    URL 'HTTP://localhost:8082/dbname/SampleWebService'  
    CLIENTPORT '5050-5060,5070';
```

It is recommended that you specify a range of port numbers when required. Only one connection is maintained at a time when you specify a single port number; the client application attempts to access all specified port numbers until it finds one to bind to. After closing the connection, a timeout period of several minutes is initiated so that no new connection can be made to the same server and port.

This feature is similar to setting the `ClientPort` network protocol option.

HTTP Request Header Management

HTTP request headers can be added, changed, or removed with the `HEADER` clause of the `CREATE PROCEDURE` and `CREATE FUNCTION` statements. You suppress an HTTP request header by referencing the name. You add or change an HTTP request header value by placing a colon after the header name following by the value. Header value specifications are optional.

For example, execute the following SQL statement in the web client database to create a procedure named `SomeOperation2` that sends requests to the specified URL that puts restrictions on HTTP request headers:

```
CREATE PROCEDURE SomeOperation2()  
    URL 'HTTP://localhost:8082/dbname/SampleWebService'  
    TYPE 'HTTP:GET'  
    HEADER 'SOAPAction\nDate\nFrom:\nCustomAlias:John Doe';
```

In this example, the `Date` header, which is automatically generated by SAP Sybase IQ, is suppressed. The `From` header is included but is not assigned a value. A new header named `CustomAlias` is included in the HTTP request and is assigned the value of `John Doe`. The GET request looks similar to the following:

```
GET /dbname/SampleWebService HTTP/1.0  
ASA-Id: e88a416e24154682bf81694feaf03052  
User-Agent: SybaseIQ/16.0.0.3600  
Accept-Charset: windows-1252, UTF-8, *  
From:  
Host: localhost:8082
```

```
Connection: close
CustomAlias: John Doe
```

Folding of long header values is supported, provided that one or more white spaces immediately follow the \n.

The following example illustrates long header value support:

```
CREATE PROCEDURE SomeOperation3 ()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:POST'
    HEADER 'heading1: This long value\n is really long for a header.
\n
    heading2:shortvalue';
```

The POST request looks similar to the following:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SybaseIQ/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:26:04 GMT
heading1: This long value is really long for a header.
heading2:shortvalue
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 0
```

Note: You must set the SOAPAction HTTP request header to the given SOAP service URI as specified in the WSDL when creating a SOAP function or procedure.

Automatically generated HTTP request headers

Modifying automatically generated headers can have unexpected results. The following HTTP request headers should not be modified without precaution:

HTTP header	Description
Accept-Charset	Always automatically generated. Changing or deleting this header may result in unexpected data conversion errors.
ASA-Id	Always automatically generated. This header ensures that the client application does not connect to itself to prevent deadlock.

HTTP header	Description
Authorization	Automatically generated when URL contains credentials. Changing or deleting this header may result in failure of the request. Only BASIC authorization is supported. User and password information should only be included when connecting via HTTPS.
Connection	Connection: close, is always automatically generated. Client applications do not support persistent connections. The connection could hang if changed.
Host	Always automatically generated. HTTP/1.1 servers are required to respond with 400 Bad Request if an HTTP/1.1 client does not provide a Host header.
Transfer-Encoding	Automatically generated when posting a request in chunk mode. Removing this header or deleting the chunked value will result in failure when the client is using CHUNK mode.
Content-Length	Automatically generated when posting a request and not in chunk mode. This header is required to tell the server the content length of the body. If the content length is wrong the connection may hang or data loss could occur.

SOAP Request Header Management

A SOAP request header is an XML fragment within a SOAP envelope. While the SOAP operation and its parameters can be thought of as an RPC (Remote Procedure Call), a SOAP request header can be used to transfer meta information within a specific request or response. SOAP request headers transport application metadata such as authorization or session criteria.

The value of a SOAPHEADER clause must be a valid XML fragment that conforms to a SOAP request header entry. Multiple SOAP request header entries can be specified. The stored procedure or function automatically injects the SOAP request header entries within a SOAP header element (SOAP-ENV:Header). SOAPHEADER values specify SOAP headers that can be declared as a static constant, or dynamically set using the parameter substitution mechanism. The following is a fragment from a sample SOAP request. It contains two XML headers called Authentication and Session respectively.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```

xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:m="HTTP://localhost:8082">
<SOAP-ENV:Header>
  <Authentication xmlns="CustomerOrderURN">
    <userName pwd="none" mustUnderstand="1">
      <first>John</first>
      <last>Smith</last>
    </userName>
  </Authentication>
  <Session xmlns="SomeSession">123456789</Session>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <m:SoapOperation>
    <m:intVariable>123</m:intVariable>
    <m:charVariable>data</m:charVariable>
  </m:SoapOperation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Processing SOAP response headers (returned by the SOAP call) differs for functions and procedures. When using a function, which is the most flexible and recommended approach, the entire SOAP response envelope is received. The response envelope can then be processed using the OPENXML operator to extract SOAP header and SOAP body data. When using a procedure, SOAP response headers can only be extracted through the use of a substitution parameter that maps to an IN or INOUT variable. A SOAP procedure allows for a maximum of one IN or INOUT parameter.

A web service function must parse the response SOAP envelope to obtain the header entries.

Examples

The following examples illustrate how to create SOAP procedures and functions that send parameters and SOAP headers. Wrapper procedures are used to populate the web service procedure calls and process the responses. The `soapAddItemProc` procedure illustrates the use of a SOAP web service procedure, the `soapAddItemFunc` function illustrates the use of a SOAP web service function, and the `httpAddItemFunc` function illustrates how a SOAP payload may be passed to an HTTP web service procedure.

The following example illustrates a SOAP client procedure that uses substitution parameters to send SOAP headers. A single INOUT parameter is used to receive SOAP headers. A wrapper stored procedure `addItemProcWrapper` that calls `soapAddItemProc` demonstrates how to send and receive soap headers including parameters.

```

CREATE PROCEDURE soapAddItemProc(amount INT, item LONG VARCHAR,
    INOUT inoutheader LONG VARCHAR, IN inheader LONG VARCHAR)
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP( OP=addItems )'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inoutheader!inheader';

CREATE PROCEDURE addItemProcWrapper(amount INT, item LONG VARCHAR,
    first_name LONG VARCHAR, last_name LONG VARCHAR)

```

```

BEGIN
    DECLARE io_header LONG VARCHAR;      // inout (write/read) soap
header
    DECLARE resxml LONG VARCHAR;
    DECLARE soap_header_sent LONG VARCHAR;
    DECLARE i_header LONG VARCHAR;      // in (write) only soap header
    DECLARE err int;
    DECLARE crsr CURSOR FOR
        CALL soapAddItemProc( amount, item, io_header, i_header );

    SET io_header = XMLELEMENT( 'Authentication',
        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
        XMLELEMENT('userName', XMLATTRIBUTES(
            'none' as pwd,
            '1' as mustUnderstand ),
            XMLELEMENT( 'first', first_name ),
            XMLELEMENT( 'last', last_name ) ) );
    SET i_header = '<Session xmlns="SomeSession">123456789</
Session>';
    SET soap_header_sent = io_header || i_header;
    OPEN crsr;
    FETCH crsr INTO resxml, err;
    CLOSE crsr;

    SELECT resxml, err, soap_header_sent, io_header AS
soap_header_received;
END;

/* example call to addItemProcWrapper */
CALL addItemProcWrapper( 5, 'shirt', 'John', 'Smith' );

```

The following example illustrates a SOAP client function that uses substitution parameters to send SOAP headers. An entire SOAP response envelope is returned. SOAP headers can be parsed using the OPENXML operator. A wrapper function addItemFuncWrapper that calls soapAddItemFunc demonstrates how to send and receive soap headers including parameters. It also shows how to process the response using the OPENXML operator.

```

CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR,
    IN inheader1 LONG VARCHAR, IN inheader2 LONG VARCHAR )
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inheader1!inheader2';

CREATE PROCEDURE addItemFuncWrapper(amount INT, item LONG VARCHAR,
    first_name LONG VARCHAR, last_name LONG VARCHAR )
BEGIN
    DECLARE i_header1 LONG VARCHAR;
    DECLARE i_header2 LONG VARCHAR;
    DECLARE res LONG VARCHAR;
    DECLARE ns LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE header_entry LONG VARCHAR;
    DECLARE localname LONG VARCHAR;

```



```

DECLARE namespaceuri LONG VARCHAR;
DECLARE r_quantity int;
DECLARE r_item LONG VARCHAR;
DECLARE r_status LONG VARCHAR;

SET i_header1 = XMLELEMENT( 'Authentication',
                           XMLATTRIBUTES('CustomerOrderURN' as xmlns),
                           XMLELEMENT('userName', XMLATTRIBUTES(
                                       'none' as pwd,
                                       '1' as mustUnderstand ),
                                       XMLELEMENT( 'first', first_name ),
                                       XMLELEMENT( 'last', last_name ) ) );
SET i_header2 = '<Session xmlns="SessionURN">123456789</
Session>';

SET res = soapAddItemFunc( amount, item, i_header1, i_header2 );

SET ns = '<ns xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"'
        || ' xmlns:mp="urn:ianywhere-com:sa-xpath-metaprop"'
        || ' xmlns:customer="CustomerOrderURN"'
        || ' xmlns:session="SessionURN"'
        || ' xmlns:tns="http://localhost:8082"></ns>';

// Process headers...
SET xpath = '//SOAP-ENV:Header/*';
BEGIN
    DECLARE crsr CURSOR FOR SELECT * FROM
        OPENXML( res, xpath, 1, ns )
        WITH ( "header_entry" LONG VARCHAR '@mp:xmltext',
              "localname"      LONG VARCHAR
              "@mp:localname",
              "namespaceuri" LONG VARCHAR
              '@mp:namespaceuri' );
    OPEN crsr;
    FETCH crsr INTO "header_entry", "localname", "namespaceuri";
    CLOSE crsr;
END;

// Process body...
SET xpath = '//tns:row';
BEGIN
    DECLARE crsr1 CURSOR FOR SELECT * FROM
        OPENXML( res, xpath, 1, ns )
        WITH ( "r_quantity" INT 'tns:quantity/text()',
              "r_item"      LONG VARCHAR 'tns:item/
text()',
              "r_status"   LONG VARCHAR 'tns:status/
text()' );
    OPEN crsr1;
    FETCH crsr1 INTO "r_quantity", "r_item", "r_status";
    CLOSE crsr1;
END;

SELECT r_item, r_quantity, r_status, header_entry, localname,
namespaceuri;

```

```
END;

/* example call to addItemFuncWrapper */
CALL addItemFuncWrapper( 6, 'shorts', 'Jack', 'Smith' );
```

The following example demonstrates how an HTTP:POST can be used as a transport for an entire SOAP payload. Rather than creating a webservice client SOAP procedure, this approach creates a webservice HTTP procedure that transports the SOAP payload. A wrapper procedure `addItemHttpWrapper` calls `httpAddItemFunc` to demonstrate the use of the POST function. It shows how to send and receive soap headers including parameters and how to accept the response.

```
CREATE FUNCTION httpAddItemFunc (soapPayload XML)
  RETURNS XML
  URL 'http://localhost:8082/itemStore'
  TYPE 'HTTP:POST:text/xml'
  HEADER 'SOAPAction: "http://localhost:8082/addItems"';

CREATE PROCEDURE addItemHttpWrapper (amount INT, item LONG VARCHAR)
  RESULT (response XML)
  BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
    '<?xml version="1.0"?>
    <SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:m="http://localhost:8082">
      <SOAP-ENV:Body>
        <m:addItems>
          <m:amount>' || amount || '</m:amount>
          <m:item>' || item || '</m:item>
        </m:addItems>
      </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc ( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
  END;

/* example call to addItemHttpWrapper */
CALL addItemHttpWrapper( 7, 'socks' );
```

Limitations

Server side SOAP services cannot currently define input and output SOAP header requirements. Therefore SOAP header metadata is not available in the WSDL output of a DISH service. A SOAP client toolkit cannot automatically generate SOAP header interfaces for an SAP Sybase IQ SOAP service endpoint.

Soap header faults are not supported.

SOAP Namespace URI Requirement

The namespace URI specifies the XML namespace used to compose the SOAP request envelope for the given SOAP operation. The domain component from URL clause is used when the namespace URI is not defined.

The server-side SOAP processor uses this URI to understand the names of the various entities in the message body of the request. The NAMESPACE clause of the CREATE PROCEDURE and CREATE FUNCTION statements specifies the namespace URI.

You may be required to specify a namespace URI before procedure calls succeed. This information is usually explained the public web server documentation, but you can obtain the required namespace URI from the WSDL available from the web server. You can generate a WSDL by accessing the DISH service if you are trying to communicate with an SAP Sybase IQ web server.

Generally, the NAMESPACE can be copied from the `targetNamespace` attribute specified at the beginning of the WSDL document within the `wsdl:definition` element. Be careful when including any trailing '/', as they are significant. Secondly, check for a `soapAction` attribute for the given SOAP operation. It should correspond to the SOAPAction HTTP header that would be generated as explained in the following paragraphs.

The NAMESPACE clause fulfills two functions. It specifies the namespace for the body of the SOAP envelope, and, if the procedure has TYPE 'SOAP:DOC' specified, it is used as the domain component of the SOAPAction HTTP header.

The following example illustrates the use of the NAMESPACE clause:

```
CREATE FUNCTION an_operation(a_parameter LONG VARCHAR)
  RETURNS LONG VARCHAR
  URL 'http://wsdl.domain.com/fictitious.asmx'
  TYPE 'SOAP:DOC'
  NAMESPACE 'http://wsdl.domain.com/'
```

Execute the following SQL statement in Interactive SQL:

```
SELECT an_operation('a_value');
```

The statement generates a SOAP request similar to the following output:

```
POST /fictitious.asmx HTTP/1.0
SOAPAction: "http://wsdl.domain.com/an_operation"
Host: wsdl.domain.com
Content-Type: text/xml
Content-Length: 387
Connection: close

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:m="http://wsdl.domain.com/">
<SOAP-ENV:Body>
  <m:an_operation>
    <m:a_parameter>a_value</m:a_parameter>
  </m:an_operation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The namespace for the prefix 'm' is set to `http://wsdl.domain.com/` and the `SOAPAction` HTTP header specifies a fully qualified URL for the SOAP operation.

The trailing slash is not a requirement for correct operation of SAP Sybase IQ but it can cause a response failure that is difficult to diagnose. The `SOAPAction` HTTP header is correctly generated regardless of the trailing slash.

When a `NAMESPACE` is not specified, the domain component from the URL clause is used as the namespace for the SOAP body, and if the procedure is of `TYPE 'SOAP:DOC'`, it is used to generate the `HTTP SOAPAction` HTTP header. If in the above example the `NAMESPACE` clause is omitted, then `http://wsdl.domain.com` is used as the namespace. The subtle difference is that a trailing slash '/' is not present. Every other aspect of the SOAP request, including the `SOAPAction` HTTP header would be identical to the above example.

The `NAMESPACE` clause is used to specify the namespace for the SOAP body as described for the `SOAP:DOC` case above. However, the `SOAPAction` HTTP header is generated with an empty value: `SOAPAction: ""`

When using the `SOAP:DOC` request type, the namespace is also used to compose the `SOAPAction` HTTP header.

Web Client SQL Statements

The following SQL statements are available to assist with web client development:

Web client related SQL statements	Description
CREATE FUNCTION statement [Web service]	Creates a web client function that makes an HTTP or SOAP over HTTP request.
ALTER FUNCTION statement	Modifies a function.
CREATE PROCEDURE statement [Web service]	Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.
ALTER PROCEDURE statement	Modifies a procedure.

Variables Supplied to Web Services

Variables can be supplied to a web service in various ways depending on the web service type.

Web client applications can supply variables to general HTTP web services using any of the following approaches:

- The suffix of the URL
- The body of an HTTP request

Variables can be supplied to the SOAP service type by including them as part of a standard SOAP envelope.

Variables Supplied in the URLs to Web Services

The HTTP web server can manage variables supplied in the URL by web browsers. These variables can be expressed in any of the following conventions:

- Appending them to the end of the URL while dividing each parameter value with a slash (/), such as in the following example:

```
http://localhost/database-name/param1/param2/param3
```

- Defining them explicitly in a URL parameter list, such as in the following example:

```
http://localhost/database-name/?
arg1=param1&arg2=param2&arg3=param3
```

- A combination of appending them to the URL and defining them in a parameter list, such as in the following example:

```
http://localhost/database-name/param4/param5?
arg1=param1&arg2=param2&arg3=param3
```

The web server interpretation of the URL depends on how the web service URL clause is specified.

Variables Supplied in the Body HTTP Requests

You can supply variables in the body of an HTTP request by specifying HTTP:POST in the TYPE clause in a web client function or procedure.

By default TYPE HTTP:POST uses application/x-www-form-urlencoded mime type. All parameters are urlencoded and passed within the body of the request. Optionally, if a media type is provided, the request Content-Type header is automatically adjusted to the provided media type and a single parameter value is uploaded within the body of the request.

Example

The following example assumes that a web service named XMLService exists on a localhost web server. Set up an SAP Sybase IQ client database, connect to it through Interactive SQL, and execute the following SQL statement:

```
CREATE PROCEDURE SendXMLContent(xmlcode LONG VARCHAR)
  URL 'http://localhost/XMLService'
  TYPE 'HTTP:POST:text/xml';
```

The statement creates a procedure that allows you to send a variable in the body of an HTTP request in text/xml format.

Execute the following SQL statement in Interactive SQL to send an HTTP request to the XMLService web service:

```
CALL SendXMLContent('<title>Hello World!</title>');
```

The procedure call assigns a value to the `xmlcode` parameter and sends it to web service.

Variables Supplied in SOAP Envelopes

You can supply variables in a SOAP envelope using the SET SOAP option of a web client function or procedure to set a SOAP operation.

The following code illustrates how to set a SOAP operation in a web client function:

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR)
  RETURNS XML
  URL 'http://localhost:8082/itemStore'
  SET 'SOAP(OP=addItems)'
  TYPE 'SOAP:DOC';
```

In this example, the `addItems` is the SOAP operation that contains the `amount` and `item` values, which are passed as parameters to the `soapAddItemFunc` function.

You can send a request by running the following sample script:

```
SELECT soapAddItemFunc(5, 'shirt');
```

A call to the `soapAddItemFunc` function call generates a SOAP envelope that looks similar to the following:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>5</m:amount>
      <m:item>shirt</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As an alternative to the previous approach, you can create your own SOAP payload and send it to the server in an HTTP wrapper.

Variables to SOAP services must be included as part of a standard SOAP request. Values supplied using other methods are ignored.

The following code illustrates how to create an HTTP wrapper procedure that builds a customized SOAP envelope:

```
CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:m="http://localhost:8082">
<SOAP-ENV:Body>
<m:addItems>
<m:amount>' || amount || '</m:amount>
<m:item>' || item || '</m:item>
</m:addItems>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
END;
```

The following code illustrates the web client function used to send the request:

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
RETURNS XML
URL 'http://localhost:8082/itemStore'
TYPE 'HTTP:POST:text/xml'
HEADER 'SOAPAction: "http://localhost:8082/addItems"';
```

You can send a request by running the following sample script:

```
CALL addItemHttpWrapper( 7, 'socks' );
```

Variables Accessed from Result Sets

Web service client calls can be made with stored functions or procedures. If made from a function, the return type must be of a character data type, such as CHAR, VARCHAR, or LONG VARCHAR. The body of the HTTP response is the returned value. No header information is included. Additional information about the request, including the HTTP status information, is returned by procedures. So, procedures are preferred when access to additional information is desired.

SOAP procedures

The response from a SOAP function is an XML document that contains the SOAP response.

SOAP responses are structured XML documents, so SAP Sybase IQ, by default, attempts to exploit this information and construct a more useful result set. Each of the top-level tags within

the returned response document is extracted and used as a column name. The contents below each of these tags in the subtree is used as the row value for that column.

For example, SAP Sybase IQ would construct the shown data set given the following SOAP response:

```
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ElizaResponse xmlns:SOAPSDK4="SoapInterop">
      <Eliza>Hi, I'm Eliza. Nice to meet you.</Eliza>
    </ElizaResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Eliza
Hi, I'm Eliza. Nice to meet you.

In this example, the response document is delimited by the `<ElizaResponse>` tags that appear within the `<SOAP-ENV:Body>` tags.

Result sets have as many columns as there are top-level tags. This result set only has one column because there is only one top-level tag in the SOAP response. This single top-level tag, Eliza, becomes the name of the column.

XML processing facilities

Information within XML result sets, including SOAP responses, can be accessed using the OPENXML procedure.

The following example uses the OPENXML procedure to extract portions of a SOAP response. This example uses a web service to expose the contents of the SYSWEBSERVICE table as a SOAP service:

```
CREATE SERVICE get_webservices
  TYPE 'SOAP'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT * FROM SYSWEBSERVICE;
```

The following web client function, which must be created in a second SAP Sybase IQ database, issues a call to this web service. The return value of this function is the entire SOAP response document. The response is in the .NET DataSet format because DNET is the default SOAP service format.

```
CREATE FUNCTION get_webservices()
  RETURNS LONG VARCHAR
  URL 'HTTP://localhost/get_webservices'
  TYPE 'SOAP:DOC';
```


The following statement illustrates how you can use the OPENXML procedure to extract two columns of the result set. The `service_name` and `secure_required` columns indicate which SOAP services are secure and where HTTPS is required.

```
SELECT *
FROM OPENXML( get_webservices(), '//row' )
WITH ( "Name" CHAR(128) 'service_name',
      "Secure?" CHAR(1) 'secure_required' );
```

This statement works by selecting the decedents of the `row` node. The `WITH` clause constructs the result set based on the two elements of interest. Assuming only the `get_webservices` web service exists, this function returns the following result set:

Name	Secure?
get_webservices	N

Result Set Retrieval from a Web Service

Web service procedures of type HTTP return all the information about a response in a two-column result set. This result set includes the response status, header information and body. The first column, is named `Attribute` and the second is named `Value`. Both are of data type LONG VARCHAR.

The result set has one row for each of the response header fields, and a row for the HTTP status line (Status attribute) and a row for the response body (Body attribute).

The following example represents a typical response:

Attribute	Value
Status	HTTP /1.0 200 OK
Body	<!DOCTYPE HTML ...><HTML>... </HTML>
Content-Type	text/html
Server	GWS/2.1
Content-Length	2234
Date	Mon, 18 Oct 2004, 16:00:00 GMT

Create the following web service stored procedure to use as an example.

```
CREATE OR REPLACE PROCEDURE SybaseWebPage ()
URL 'http://www.sybase.com/mobilize'
TYPE 'HTTP';
```

Execute the following SELECT query to obtain the response from the web service as a result set.

```
SELECT * FROM SybaseWebPage()
WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

Because the web service procedure does not describe the shape of the result set, the WITH clause is required to define a temporary view.

The results of a query can be stored in a table. Execute the following SQL statement to create a table to contain the values of the result set.

```
CREATE TABLE StoredResults(  
    Attribute LONG VARCHAR,  
    Value     LONG VARCHAR  
);
```

The result set can be inserted into the `StoredResults` table as follows:

```
INSERT INTO StoredResults  
    SELECT * FROM SybaseWebPage()  
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

You can add clauses according to the usual syntax of the SELECT statement. For example, if you want only a specific row of the result set you can add a WHERE clause to limit the results of the SELECT to only one row.

```
SELECT * FROM SybaseWebPage()  
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR)  
    WHERE Attribute = 'Status';
```

This SELECT statement retrieves only the status information from the result set. It can be used to verify that the call was successful.

SOAP Data Types

By default, the XML encoding of parameter input is string and the result set output for SOAP service formats contains no information that specifically describes the data type of the columns in the result set. For all formats, parameter data types are string. For the DNET format, within the schema section of the response, all columns are typed as string. CONCRETE and XML formats contain no data type information in the response. This default behavior can be manipulated using the DATATYPE clause.

SAP Sybase IQ enables data typing using the DATATYPE clause. Data type information can be included in the XML encoding of parameter input and result set output or responses for all SOAP service formats. This simplifies parameter passing from SOAP toolkits by not requiring client code to explicitly convert parameters to Strings. For example, an integer can be passed as an int. XML encoded data types enable a SOAP toolkit to parse and cast the data to the appropriate type.

When using string data types exclusively, the application needs to implicitly know the data type for every column within the result set. This is not necessary when data typing is requested of the web server. To control whether data type information is included, the DATATYPE clause can be used when the web service is defined.

Here is an example of a web service definition that enlists data typing for the result set response.

```
CREATE SERVICE "SASoapTest/EmployeeList"
  TYPE 'SOAP'
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA
  DATATYPE OUT
  AS SELECT * FROM Employees;
```

In this example, data type information is requested for result set responses only since this service does not have parameters.

Data typing is applicable to all SAP Sybase IQ web services defined as type 'SOAP'.

Data typing of input parameters

Data typing of input parameters is supported by simply exposing the parameter data types as their true data types in the WSDL generated by the DISH service.

A typical string parameter definition (or a non-typed parameter) would look like the following:

```
<s:element minOccurs="0" maxOccurs="1" name="a_varchar"
nillable="true" type="s:string" />
```

The String parameter may be nillable, that is, it may or may not occur.

For a typed parameter such as an integer, the parameter must occur and is not nillable. The following is an example.

```
<s:element minOccurs="1" maxOccurs="1" name="an_int"
nillable="false" type="s:int" />
```

Data typing of output parameters

All SAP Sybase IQ web services of type 'SOAP' may expose data type information within the response data. The data types are exposed as attributes within the rowset column element.

The following is an example of a typed SimpleDataSet response from a SOAP FORMAT 'CONCRETE' web service.

```
<SOAP-ENV:Body>
  <tns:test_types_concrete_onResponse>
    <tns:test_types_concrete_onResult xsi:type='tns:SimpleDataset'>
      <tns:rowset>
        <tns:row>
          <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
          <tns:i xsi:type="xsd:int">99</tns:i>
          <tns:ii xsi:type="xsd:long">99999999</tns:ii>
          <tns:f xsi:type="xsd:float">3.25</tns:f>
          <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
          <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
          <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
        </tns:row>
      </tns:rowset>
    </tns:test_types_concrete_onResult>
  <tns:sqlcode>0</tns:sqlcode>
```

```
</tns:test_types_concrete_onResponse>
</SOAP-ENV:Body>
```

The following is an example of a response from a SOAP FORMAT 'XML' web service returning the XML data as a string. The interior rowset consists of encoded XML and is presented here in its decoded form for legibility.

```
<SOAP-ENV:Body>
  <tns:test_types_XML_onResponse>
    <tns:test_types_XML_onResult xsi:type='xsd:string'>
      <tns:rowset
        xmlns:tns="http://localhost/satest/dish"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <tns:row>
          <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
          <tns:i xsi:type="xsd:int">99</tns:i>
          <tns:ii xsi:type="xsd:long">99999999</tns:ii>
          <tns:f xsi:type="xsd:float">3.25</tns:f>
          <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
          <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
          <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
        </tns:row>
      </tns:rowset>
    </tns:test_types_XML_onResult>
    <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_XML_onResponse>
</SOAP-ENV:Body>
```

In addition to the data type information, the namespace for the elements and the XML schema provides all the information necessary for post processing by an XML parser. When no data type information exists in the result set (DATATYPE OFF or IN) then the xsi:type and the XML schema namespace declarations are omitted.

An example of a SOAP FORMAT 'DNET' web service returning a typed SimpleDataSet follows:

```
<SOAP-ENV:Body>
  <tns:test_types_dnet_outResponse>
    <tns:test_types_dnet_outResult
      xsi:type='sqlresultstream:SqlRowSet'>
      <xsd:schema id='Schema2'
        xmlns:xsd='http://www.w3.org/2001/XMLSchema'
        xmlns:msdata='urn:schemas-microsoft.com:xml-msdata'>
        <xsd:element name='rowset' msdata:IsDataSet='true'>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name='row' minOccurs='0' maxOccurs='unbounded'>
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name='lvc' minOccurs='0' type='xsd:string' />
                    <xsd:element name='ub' minOccurs='0'
                      type='xsd:unsignedByte' />
                    <xsd:element name='s' minOccurs='0' type='xsd:short' />
                    <xsd:element name='us' minOccurs='0'
```

```

type='xsd:unsignedShort' />
  <xsd:element name='i' minOccurs='0' type='xsd:int' />
  <xsd:element name='ui' minOccurs='0'
type='xsd:unsignedInt' />
  <xsd:element name='l' minOccurs='0' type='xsd:long' />
  <xsd:element name='ul' minOccurs='0'
type='xsd:unsignedLong' />
  <xsd:element name='f' minOccurs='0' type='xsd:float' />
  <xsd:element name='d' minOccurs='0' type='xsd:double' />
  <xsd:element name='bin' minOccurs='0'
type='xsd:base64Binary' />
  <xsd:element name='bool' minOccurs='0'
type='xsd:boolean' />
  <xsd:element name='num' minOccurs='0' type='xsd:decimal' />
  <xsd:element name='dc' minOccurs='0' type='xsd:decimal' />
  <xsd:element name='date' minOccurs='0' type='xsd:date' />
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<diffgr:diffgram xmlns:msdata='urn:schemas-microsoft-com:xml-
msdata' xmlns:diffgr='urn:schemas-microsoft-com:xml-diffgram-v1'>
  <rowset>
    <row>
      <lvc>Hello World</lvc>
      <ub>128</ub>
      <s>-99</s>
      <us>33000</us>
      <i>-2147483640</i>
      <ui>4294967295</ui>
      <l>-9223372036854775807</l>
      <ul>18446744073709551615</ul>
      <f>3.25</f>
      <d>.5555555555555555582</d>
      <bin>QUJD</bin>
      <bool>1</bool>
      <num>123456.123457</num>
      <dc>-1.756000</dc>
      <date>2006-05-29-04:00</date>
    </row>
  </rowset>
</diffgr:diffgram>
</tns:test_types_dnet_outResult>
<tns:sqlcode>0</tns:sqlcode>
</tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>

```

Mapping SAP Sybase IQ types to XML schema types

SAP Sybase IQ type	XML schema type	XML example
CHAR	string	Hello World

SAP Sybase IQ type	XML schema type	XML example
VARCHAR	string	Hello World
LONG VARCHAR	string	Hello World
TEXT	string	Hello World
NCHAR	string	Hello World
NVARCHAR	string	Hello World
LONG NVARCHAR	string	Hello World
NTEXT	string	Hello World
UNIQUEIDENTIFIER	string	12345678-1234-5678-9012-123456789012
UNIQUEIDENTIFIERSTR	string	12345678-1234-5678-9012-123456789012
XML	This is user defined. A parameter is assumed to be valid XML representing a complex type (for example, base64Binary, SOAP array, struct).	<inputHexBinary xsi:type="xsd:hexBinary">414243 </inputHexBinary> (interpreted as 'ABC')
BIGINT	long	-9223372036854775807
UNSIGNED BIGINT	unsignedLong	18446744073709551615
BIT	boolean	1
VARBIT	string	11111111
LONG VARBIT	string	00000000000000001000000000000000
DECIMAL	decimal	-1.756000
DOUBLE	double	.55555555555555582
FLOAT	float	12.3456792831420898
INTEGER	int	-2147483640
UNSIGNED INTEGER	unsignedInt	4294967295
NUMERIC	decimal	123456.123457
REAL	float	3.25

SAP Sybase IQ type	XML schema type	XML example
SMALLINT	short	-99
UNSIGNED SMALLINT	unsignedShort	33000
TINYINT	unsignedByte	128
MONEY	decimal	12345678.9900
SMALLMONEY	decimal	12.3400
DATE	date	2006-11-21-05:00
DATETIME	dateTime	2006-05-21T09:00:00.000-08:00
SMALLDATETIME	dateTime	2007-01-15T09:00:00.000-08:00
TIME	time	14:14:48.980-05:00
TIMESTAMP	dateTime	2007-01-12T21:02:14.420-06:00
TIMESTAMP WITH TIME ZONE	dateTime	2007-01-12T21:02:14.420-06:00
BINARY	base64Binary	AAAAZg==
IMAGE	base64Binary	AAAAZg==
LONG BINARY	base64Binary	AAAAZg==
VARBINARY	base64Binary	AAAAZg==

When one or more parameters are of type NCHAR, NVARCHAR, LONG NVARCHAR, or NTEXT then the response output is in UTF8. If the client database uses the UTF-8 character encoding, there is no change in behavior (since NCHAR and CHAR data types are the same). However, if the database does not use the UTF-8 character encoding, then all parameters that are not an NCHAR data type are converted to UTF8. The value of the XML declaration encoding and Content-Type HTTP header will correspond to the character encoding used.

Mapping XML schema types to Java types

XML schema type	Java data type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger

XML schema type	Java data type
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

SOAP Structured Data Types

The SAP Sybase IQ server as a web service client may interface to a web service using a function or a procedure.

XML Return Values

A string representation within a result set may suffice for simple return data types. The use of a stored procedure may be warranted in this case.

The use of web service functions are a better choice when returning complex data such as arrays or structures. For function declarations, the RETURN clause can specify an XML data type. The returned XML can be parsed using OPENXML to extract the elements of interest.

A return of XML data such as dateTime is rendered within the result set verbatim. For example, if a TIMESTAMP column was included within a result set, it would be formatted as an XML dateTime string (2006-12-25T12:00:00.000-05:00) not as a string (2006-12-25 12:00:00.000).

XML Parameter Values

The SAP Sybase IQ XML data type is supported for use as a parameter within web service functions and procedures. For simple types, the parameter element is automatically constructed when generating the SOAP request body. However, for XML parameter types, this cannot be done since the XML representation of the element may require attributes that provide additional data. Therefore, when generating the XML for a parameter whose data type is XML, the root element name must correspond to the parameter name.

```
<inputHexBinary xsi:type="xsd:hexBinary">414243</inputHexBinary>
```

The XML type demonstrates how to send a parameter as a hexBinary XML type. The SOAP endpoint expects that the parameter name (or in XML terms, the root element name) is "inputHexBinary".

Cookbook Constants

Knowledge of how SAP Sybase IQ references namespaces is required to construct complex structures and arrays. The prefixes listed here correspond to the namespace declarations generated for an SAP Sybase IQ SOAP request envelope.

SAP Sybase IQ XML Prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
m	namespace as defined in the NAMESPACE clause

Complex Data Type Examples

The following three examples demonstrate how to create web service client functions taking parameters that represent an array, a structure, and an array of structures. The web service functions will communicate to SOAP operations (or RPC function names) named echoFloatArray, echoStruct, and echoStructArray respectively. The common namespace used for Interoperability testing is http://soapinterop.org/, allowing a given function to test against alternative Interoperability servers simply by changing the URL clause to the chosen SOAP endpoint.

The examples are designed to issue requests to the Microsoft SOAP ToolKit 3.0 Round 2 Interoperability test server at <http://mssoapinterop.org/stkV3>.

All the examples use a table to generate the XML data. The following shows how to set up that table.

```
CREATE LOCAL TEMPORARY TABLE SoapData
(
    seqno INT DEFAULT AUTOINCREMENT,
    i INT,
    f FLOAT,
    s LONG VARCHAR
) ON COMMIT PRESERVE ROWS;

INSERT INTO SoapData (i,f,s)
VALUES (99,99.999,'Ninety-Nine');

INSERT INTO SoapData (i,f,s)
VALUES (199,199.999,'Hundred and Ninety-Nine');
```

The following three functions send SOAP requests to the Interoperability server. This sample issues requests to the Microsoft Interop server:

```
CREATE FUNCTION echoFloatArray(inputFloatArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStruct(inputStruct XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStructArray(inputStructArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/'
NAMESPACE 'http://soapinterop.org/';
```

Finally, the three example statements along with the XML representation of their parameters are presented:

1. The parameters in the following example represent an array.

```
SELECT echoFloatArray(
    XMLELEMENT( 'inputFloatArray',
        XMLATTRIBUTES( 'xsd:float[2]' as "SOAP-ENC:arrayType" ),
        (
            SELECT XMLAGG( XMLELEMENT( 'number', f ) ORDER BY seqno )
            FROM SoapData
        )
    )
);
```

The stored procedure echoFloatArray will send the following XML to the Interoperability server.

```
<inputFloatArray SOAP-ENC:arrayType="xsd:float[2]">
<number>99.9990005493164</number>
<number>199.998992919922</number>
</inputFloatArray>
```

The response from the Interoperability server is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
    encoding/">
    <SOAPSDK4:echoFloatArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/"
      <Result SOAPSDK3:arrayType="SOAPSDK1:float[2]"
        SOAPSDK3:offset="[0]"
        SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK3:float>99.9990005493164</SOAPSDK3:float>
        <SOAPSDK3:float>199.998992919922</SOAPSDK3:float>
      </Result>
    </SOAPSDK4:echoFloatArrayResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the response was stored in a variable, then it can be parsed using OPENXML.

```
SELECT * FROM OPENXML( resp, '/*:Result/*' )
WITH ( varFloat FLOAT 'text()' );
```

varFloat
99.9990005493
199.9989929199

2. The parameters in the following example represent a structure.

```
SELECT echoStruct(
  XMLELEMENT('inputStruct',
    (
      SELECT XMLFOREST( s as varString,
                        i as varInt,
                        f as varFloat )
      FROM SoapData
      WHERE seqno=1
    )
  )
);
```

The stored procedure echoStruct will send the following XML to the Interoperability server.

```
<inputStruct>
  <varString>Ninety-Nine</varString>
  <varInt>99</varInt>
  <varFloat>99.9990005493164</varFloat>
</inputStruct>
```

The response from the Interoperability server is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoStructResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result href="#idl"/>
    </SOAPSDK4:echoStructResponse>
    <SOAPSDK5:SOAPStruct
      xmlns:SOAPSDK5="http://soapinterop.org/xsd"
      id="idl"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK5:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK5:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the response was stored in a variable, then it can be parsed using OPENXML.

```
SELECT * FROM OPENXML( resp, '//*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493

3. The parameters in the following example represent an array of structures.

```
SELECT echoStructArray(
  XMLELEMENT( 'inputStructArray',
    XMLATTRIBUTES( 'http://soapinterop.org/xsd' AS "xmlns:q2",
      'q2:SOAPStruct[2]' AS "SOAP-ENC:arrayType" ),
    (
      SELECT XMLAGG (
        XMLElement('q2:SOAPStruct',
```

```

        XMLFOREST( s as varString,
                  i as varInt,
                  f as varFloat )
    )
    ORDER BY seqno
)
FROM SoapData
)
);

```

The stored procedure echoFloatArray will send the following XML to the Interoperability server.

```

<inputStructArray xmlns:q2="http://soapinterop.org/xsd"
  SOAP-ENC:arrayType="q2:SOAPStruct[2]">
  <q2:SOAPStruct>
    <varString>Ninety-Nine</varString>
    <varInt>99</varInt>
    <varFloat>99.9990005493164</varFloat>
  </q2:SOAPStruct>
  <q2:SOAPStruct>
    <varString>Hundred and Ninety-Nine</varString>
    <varInt>199</varInt>
    <varFloat>199.998992919922</varFloat>
  </q2:SOAPStruct>
</inputStructArray>

```

The response from the Interoperability server is shown below.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoStructArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result xmlns:SOAPSDK5="http://soapinterop.org/xsd"
        SOAPSDK3:arrayType="SOAPSDK5:SOAPStruct[2]"
        SOAPSDK3:offset="[0]" SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK5:SOAPStruct href="#id1"/>
        <SOAPSDK5:SOAPStruct href="#id2"/>
      </Result>
    </SOAPSDK4:echoStructArrayResponse>
    <SOAPSDK6:SOAPStruct
      xmlns:SOAPSDK6="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK6:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK6:SOAPStruct>

```

```
<SOAPSDK7:SOAPStruct
  xmlns:SOAPSDK7="http://soapinterop.org/xsd"
  id="id2"
  SOAPSDK3:root="0"
  SOAPSDK2:type="SOAPSDK7:SOAPStruct">
  <varString>Hundred and Ninety-Nine</varString>
  <varInt>199</varInt>
  <varFloat>199.998992919922</varFloat>
</SOAPSDK7:SOAPStruct>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the response was stored in a variable, then it can be parsed using OPENXML.

```
SELECT * FROM OPENXML( resp, '/*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493
Hundred and Ninety-Nine	199	199.9989929199

Substitution Parameters Used for Clause Values

Declared parameters to a stored procedure or function are automatically substituted for placeholders within a clause definition each time the stored procedure or function is run. Substitution parameters allow the creation of general web service procedures that dynamically configure clauses at run time. Any substrings that contain an exclamation mark '!' followed by the name of one of the declared parameters is replaced by that parameter's value. In this way one or more parameter values may be substituted to derive one or more clause values at runtime.

Parameter substitution requires adherence to the following rules:

- All parameters used for substitution must be alphanumeric. Underscores are not allowed.
- A substitution parameter must be followed immediately by a non-alphanumeric character or termination. For example, !sizeXL is not substituted with the value of a parameter named size because X is alphanumeric.
- A substitution parameter that is not matched to a parameter name is ignored.
- An exclamation mark (!) can be escaped with another exclamation mark.

For example, the following procedure illustrates the use of parameter substitution. URL and HTTP header definitions must be passed as parameters.

```
CREATE PROCEDURE test(uid CHAR(128), pwd CHAR(128), headers LONG
VARCHAR)
  URL 'http://!uid:!pwd@localhost/myservice'
  HEADER '!headers';
```

You can then use the following statement to call the `test` procedure and initiate an HTTP request:

```
CALL test('dba', 'sql', 'NewHeader1:value1\nNewHeader2:value2');
```

Different values can be used each time this procedure is called.

Encryption certificate example

You can use parameter substitution to pass encryption certificates from a file and pass them to a stored procedure or stored function.

The following example illustrates how to pass a certificate as a substitution string:

```
CREATE PROCEDURE secure(cert LONG VARCHAR)
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Root';
```

The certificate is read from a file and passed to `secure` in the following call.

```
CALL secure( xp_read_file('%ALLUSERSPROFILE%/SybaseIQ/demo\
\Certificates\rsaroot.crt) );
```

This example is for illustration only. The certificate can be read directly from a file using the `file=` keyword for the `CERTIFICATE` clause.

No matching parameter name example

Placeholders with no matching parameter name are automatically deleted.

For example, the parameter `size` would not be substituted for the placeholder in the following procedure:

```
CREATE PROCEDURE orderitem (size CHAR(18))
URL 'HTTP://localhost/salesserver/order?size=!sizeXL'
TYPE 'SOAP:RPC';
```

In this example, `!sizeXL` is always deleted because it is a placeholder for which there is no matching parameter.

Parameters can be used to replace placeholders within the body of the stored function or stored procedure at the time the function or procedure is called. If placeholders for a particular variable do not exist, the parameter and its value are passed as part of the request. Parameters and values used for substitution in this manner are not passed as part of the request.

HTTP and SOAP Request Structures

All parameters to a function or procedure, unless used during parameter substitution, are passed as part of the web service request. The format in which they are passed depends on the type of the web service request.

Parameter values that are not of character or binary data types are converted to a string representation before being added to the request. This process is equivalent to casting the value to a character type. The conversion is done in accordance with the data type formatting

option settings at the time the function or procedure is invoked. In particular, the conversion can be affected by such options as precision, scale, and timestamp_format.

HTTP request structures

Parameters for type HTTP:GET are URL encoded and placed within the URL. Parameter names are used verbatim as the name for HTTP variables. For example, the following procedure declares two parameters:

```
CREATE PROCEDURE test(a INTEGER, b CHAR(128))
  URL 'HTTP://localhost/myservice'
  TYPE 'HTTP:GET';
```

If this procedure is invoked with the two values 123 and 'xyz', then the URL used for the request is equivalent to that shown below:

```
HTTP://localhost/myservice?a=123&b=xyz
```

If the type is HTTP:POST, the parameters and their values are URL encoded and placed within the body of the request. After the headers, the following text appears in the body of the HTTP request for the two parameter and values:

```
a=123&b=xyz
```

SOAP request structures

Parameters passed to SOAP requests are bundled as part of the request body, as required by the SOAP specification:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:test>
      <m:a>123</m:a>
      <m:b>abc</m:b>
    </m:test>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

How to Log Web Client Requests

Web service client information, including HTTP requests and transport data, can be logged to the web service client log file. The web service client log file can be specified with the -zoc server option or by using the sa_server_option system procedure:

```
CALL sa_server_option( 'WebClientLogFile', 'clientinfo.txt' );
```

Logging is enabled automatically when you specify the -zoc server option. You can enable and disable logging to this file using the sa_server_option system procedure:

```
CALL sa_server_option( 'WebClientLogging', 'ON' );
```


Web Services References

This section provides information about web service references.

Web Service Error Code Reference

The HTTP server generates standard web service errors when requests fail. These errors are assigned numbers consistent with protocol standards.

The following are some typical errors that you may encounter:

Number	Name	SOAP fault	Description
301	Moved permanently	Server	The requested page has been permanently moved. The server automatically redirects the request to the new location.
304	Not Modified	Server	The server has decided, based on information in the request, that the requested data has not been modified since the last request and so it does not need to be sent again.
307	Temporary Redirect	Server	The requested page has been moved, but this change may not be permanent. The server automatically redirects the request to the new location.
400	Bad Request	Client.BadRequest	The HTTP request is incomplete or malformed.

Number	Name	SOAP fault	Description
401	Authorization Required	Client.Authorization	Authorization is required to use the service, but a valid user name and password were not supplied.
403	Forbidden	Client.Forbidden	You do not have permission to access the database.
404	Not Found	Client.NotFound	The named database is not running on the server, or the named web service does not exist.
408	Request Timeout	Server.RequestTimeout	The maximum connection idle time was exceeded while receiving the request.
411	HTTP Length Required	Client.LengthRequired	The server requires that the client include a Content-Length specification in the request. This typically occurs when uploading data to the server.
413	Entity Too Large	Server	The request exceeds the maximum permitted size.
414	URI Too Large	Server	The length of the URI exceeds the maximum allowed length.
500	Internal Server Error	Server	An internal error occurred. The request could not be processed.
501	Not Implemented	Server	The HTTP request method is not GET, HEAD, or POST.

Number	Name	SOAP fault	Description
502	Bad Gateway	Server	The document requested resides on a third-party server and the server received an error from the third-party server.
503	Service Unavailable	Server	The number of connections exceeds the allowed maximum.

Faults are returned to the client as SOAP faults as defined by the following the SOAP version 1.1 standards when a SOAP service fails:

- When an error in the application handling the request generates a `SQLCODE`, a SOAP Fault is returned with a faultcode of Client, possibly with a sub-category, such as Procedure. The faultstring element within the SOAP Fault is set to a detailed explanation of the error and a detail element contains the numeric `SQLCODE` value.
- In the event of a transport protocol error, the faultcode is set to either Client or Server, depending on the error, faultstring is set to the HTTP transport message, such as `404 Not Found`, and the detail element contains the numeric HTTP error value.
- SOAP Fault messages generated due to application errors that return a `SQLCODE` value are returned with an HTTP status of `200 OK`.

The appropriate HTTP error is returned in a generated HTML document if the client cannot be identified as a SOAP client.

HTTP Web Service Examples

Several sample implementations of web services are located in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP` directory. For more information about the samples, see `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\readme.txt`.

Tutorial: Create a Web Server and Access It from a Web Client

This tutorial illustrates how to create a web server using a SQL Anywhere database server and then send requests to it from a web client database server.

Required Software

- SAP Sybase IQ

Competencies and Experience

- Familiarity with XML
- Familiarity with MIME (Multipurpose Internet Mail Extensions) types
- Basic knowledge of SAP Sybase IQ web services

Goals

- Create and start a new SAP Sybase IQ web server database.
- Create a web service.
- Set up a procedure that returns the information contained in HTTP requests.
- Create and start a new SAP Sybase IQ web client database.
- Send HTTP:POST requests from the web client to the database server.
- Send an HTTP responses from the web server to the web client.

Privileges

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Lesson 1: Setting Up a Web Server to Receive Requests and Send Responses

The goal of this lesson is to set up an SAP Sybase IQ web server running a web service.

Prerequisites

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Create a web server and access it from a web client.

Task

1. Create an SAP Sybase IQ database that will be used to contain web service definitions.

```
iqinit -dba <user_id>,<password> echo
```
2. Start a network database server using this database. This server will act as a web server.

```
iqsrv16 -xs http(port=8082) -n echo echo.db
```

The HTTP web server is set to listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

3. Connect to the database server with Interactive SQL.

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=echo"
```

4. Create a new web service to accept incoming requests.

```
CREATE SERVICE EchoService  
TYPE 'RAW'  
USER DBA
```

```
AUTHORIZATION OFF
SECURE OFF
AS CALL Echo();
```

This statement creates a new service named `EchoService` that calls a stored procedure named `Echo` when a web client sends a request to the service. It generates an HTTP response body without any formatting (RAW) for the web client.

5. Create the `Echo` procedure to handle incoming requests.

```
CREATE OR REPLACE PROCEDURE Echo()
BEGIN
    DECLARE request_body LONG VARCHAR;
    DECLARE request_mimetype LONG VARCHAR;

    SET request_mimetype = http_header( 'Content-Type' );
    SET request_body = isnull( http_variable('text'),
http_variable('body') );
    IF request_body IS NULL THEN
        CALL sa_set_http_header('Content-Type', 'text/plain' );
        SELECT 'failed'
    ELSE
        CALL sa_set_http_header('Content-Type',
request_mimetype );
        SELECT request_body;
    END IF;
END
```

This procedure formats the `Content-Type` header and the body of the response that is sent to the web client.

A web server is set up to receive requests and send responses.

Next

Proceed to Lesson 2: Sending requests from a web client and receiving responses.

Lesson 2: Sending Requests from a Web Client and Receiving Responses

In this lesson, you set up a database client to send requests to a web server using the POST method and to receive the web server's responses.

Prerequisites

This lesson assumes that you have set up a web server as instructed in Lesson 1.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Create a web server and access it from a web client.

Task

This lesson contains several references to `localhost`. Use the host name or IP address of the web server from Lesson 1 instead of `localhost` if you are not running the web client on the same computer as the web server.

1. Create an SAP Sybase IQ database that will be used to contain web client procedures.

```
iqinit -dba <user_id>,<password> echo_client
```

2. Start a network database server using this database. This server will act as a web client.

```
iqsrv16 echo_client.db
```

3. Connect to the database server with Interactive SQL.

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=echo_client"
```

4. Create a new stored procedure to send requests to a web service.

```
CREATE OR REPLACE PROCEDURE SendWithMimeType (  
    value LONG VARCHAR,  
    mimeType LONG VARCHAR,  
    urlSpec LONG VARCHAR  
)  
URL '!urlSpec'  
TYPE 'HTTP:POST:!mimeType';
```

The `SendWithMimeType` procedure has three parameters. The `value` parameter represents the body of the request that should be sent to the web service. The `urlSpec` parameter indicates the URL to use to connect to the web service. The `mimeType` indicates which MIME type to use for the HTTP:POST.

5. Send a request to the web server and obtain the response.

```
CALL SendWithMimeType('<hello>this is xml</hello>',  
    'text/xml',  
    'http://localhost:8082/EchoService'  
);
```

The `http://localhost:8082/EchoService` string indicates that the web server runs on `localhost` and listens on port `8082`. The desired web service is named `EchoService`.

6. Try a different MIME type and observe the response.

```
CALL SendWithMimeType('{"menu": { "id": "file", "value": "File",  
"popup": {  
    "menuitem": [{"value": "New", "onclick": "CreateNew()"},  
                {"value": "Open", "onclick": "Open()"},  
                {"value": "Close", "onclick": "Close()"} ] } } }',  
    'application/json',  
    'http://localhost:8082/EchoService'  
);
```

A web client is set up to send HTTP requests to a web server using the POST method and receive the web server's response.

Tutorial: Using SAP Sybase IQ to Access a SOAP/DISH Service

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

Required Software

- SAP Sybase IQ

Competencies and Experience

- Familiarity with SOAP
- Basic knowledge of SAP Sybase IQ web services

Goals

- Create and start a new SAP Sybase IQ web server database.
- Create a SOAP web service.
- Set up a procedure that converts a client-supplied Fahrenheit value to a Celsius value.
- Create and start a new SAP Sybase IQ web client database.
- Send a SOAP request from the web client to the database server.
- Send a SOAP response from the database server to the web client.

Privileges

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses

In this lesson, you set up a new database server and create a SOAP service to handle incoming SOAP requests. The server anticipates SOAP requests that provide a Fahrenheit temperature value that is converted to the equivalent Celsius degrees.

Prerequisites

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using SAP Sybase IQ to access a SOAP/DISH service.

Task

1. Create an SAP Sybase IQ database that will be used to contain web service definitions.

```
iqinit -dba <user_id>,<password> ftc
```

2. Start a database server using this database. This server will act as a web server.

```
iqsrv16 -xs http(port=8082) -n ftc ftc.db
```

The HTTP web server is set to listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

3. Connect to the database server with Interactive SQL.

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc"
```

4. Create a new DISH service to accept incoming requests.

```
CREATE SERVICE soap_endpoint
  TYPE 'DISH'
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA;
```

This statement creates a new DISH service named `soap_endpoint` that handles incoming SOAP service requests.

5. Create a new SOAP service to handle Fahrenheit to Celsius conversions.

```
CREATE SERVICE FtoCService
  TYPE 'SOAP'
  FORMAT 'XML'
  AUTHORIZATION OFF
  USER DBA
  AS CALL FToCConverter( :fahrenheit );
```

This statement creates a new SOAP service named `FtoCService` that generates XML-formatted strings as output. It calls a stored procedure named `FToCConverter` when a web client sends a SOAP request to the service.

6. Create the `FToCConverter` procedure to handle incoming SOAP requests. This procedure performs the necessary calculations to convert a client-supplied Fahrenheit temperature value to the equivalent Celsius temperature value.

```
CREATE OR REPLACE PROCEDURE FToCConverter( temperature FLOAT )
BEGIN
  DECLARE hd_key LONG VARCHAR;
  DECLARE hd_entry LONG VARCHAR;
  DECLARE alias LONG VARCHAR;
  DECLARE first_name LONG VARCHAR;
  DECLARE last_name LONG VARCHAR;
  DECLARE xpath LONG VARCHAR;
  DECLARE authinfo LONG VARCHAR;
  DECLARE namespace LONG VARCHAR;
  DECLARE mustUnderstand LONG VARCHAR;
header_loop:
  LOOP
    SET hd_key = NEXT_SOAP_HEADER( hd_key );
    IF hd_key IS NULL THEN
      -- no more header entries
      LEAVE header_loop;
    END IF;
    IF hd_key = 'Authentication' THEN
      SET hd_entry = SOAP_HEADER( hd_key );
```



```

SET xpath = '/*:' || hd_key || '/*:userName';
SET namespace = SOAP_HEADER( hd_key, 1, '@namespace' );
SET mustUnderstand = SOAP_HEADER( hd_key, 1,
'mustUnderstand' );
BEGIN
-- parse the XML returned in the SOAP header
DECLARE crsr CURSOR FOR
SELECT * FROM OPENXML( hd_entry, xpath )
WITH ( alias LONG VARCHAR '@*:alias',
first_name LONG VARCHAR '*:first/text()',
last_name LONG VARCHAR '*:last/text()' );
OPEN crsr;
FETCH crsr INTO alias, first_name, last_name;
CLOSE crsr;
END;

-- build a response header
-- based on the pieces from the request header
SET authinfo =
XMLELEMENT( 'Authentication',
XMLATTRIBUTES(
namespace as xmlns,
alias,
mustUnderstand ),
XMLELEMENT( 'first', first_name ),
XMLELEMENT( 'last', last_name ) );
CALL SA_SET_SOAP_HEADER( 'authinfo', authinfo );
END IF;
END LOOP header_loop;
SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) AS answer;
END;

```

The `NEXT_SOAP_HEADER` function is used in a `LOOP` structure to iterate through all the header names in a SOAP request, and exits the loop when the `NEXT_SOAP_HEADER` function returns `NULL`.

Note: This function does not necessarily iterate through the headers in the order that they appear in the SOAP request.

The `SOAP_HEADER` function returns the header value or `NULL` when the header name does not exist. The `FTtoCConverter` procedure searches for a header named `Authentication` and extracts the header structure, including the `@namespace` and `mustUnderstand` attributes. The `@namespace` header attribute is a special SAP Sybase IQ attribute used to access the namespace (`xmlns`) of the given header entry.

The following is an XML string representation of a possible `Authentication` header structure, where the `@namespace` attribute has a value of `"SecretAgent"`, and `mustUnderstand` has a value of `1`:

```

<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>

```

```
</userName>  
</Authentication>
```

The OPENXML system procedure in the SELECT statement parses the XML header using the XPath string `"/*:Authentication/*:userName"` to extract the alias attribute value and the contents of the `first` and `last` tags. The result set is processed using a cursor to fetch the three column values.

At this point, you have all the information of interest that was passed to the web service. You have the temperature in Fahrenheit degrees and you have some additional attributes that were passed to the web service in a SOAP header. You could look up the name and alias that were provided to see if the person is authorized to use the web service. However, this exercise is not shown in the example.

The SET statement is used to build a SOAP response in XML format to send to the client. The following is an XML string representation of a possible SOAP response. It is based on the above Authentication header structure example.

```
<Authentication xmlns="SecretAgent" alias="99"  
mustUnderstand="1">  
  <first>Susan</first>  
  <last>Hilton</last>  
</Authentication>
```

The SA_SET_SOAP_HEADER system procedure is used to set the SOAP response header that will be sent to the client.

The final SELECT statement is used to convert the supplied Fahrenheit value to a Celsius value. This information is relayed back to the client.

At this point, you now have a running SQL Anywhere web server that provides a service for converting temperatures from degrees Fahrenheit to degrees Celsius. This service processes a SOAP header from the client and sends a SOAP response back to the client.

Next

In the next lesson, you develop an example of a client that can send SOAP requests to the web server and receive SOAP responses from the web server.

Lesson 2: Setting Up a Web Client to Send SOAP Requests and Receive SOAP Responses

In this lesson, you set up a web client that sends SOAP requests and receives SOAP responses.

Prerequisites

This lesson assumes that you have set up a web server as instructed in the previous lesson.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using SAP Sybase IQ to access a SOAP/DISH service.

Task

This lesson contains several references to `localhost`. Use the host name or IP address of the web server from lesson 1 instead of `localhost` if you are not running the web client on the same computer as the web server.

1. Run the following command to create an SAP Sybase IQ database:

```
iqinit -dba <user_id>,<password> ftc_client
```

2. Start the database client using the following command:

```
iqsrv16 ftc_client.db
```

3. Connect to the database in Interactive SQL using the following command:

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc_client"
```

4. Create a new stored procedure to send SOAP requests to a DISH service.

Execute the following SQL statement in Interactive SQL:

```
CREATE OR REPLACE PROCEDURE FtoC( fahrenheit FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR )
    URL 'http://localhost:8082/soap_endpoint'
    SET 'SOAP(OP=FtoCService)'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inoutheader!inheader!;
```

The `http://localhost:8082/soap_endpoint` string in the URL clause indicates that the web server runs on `localhost` and listens on port 8082. The desired DISH web service is named `soap_endpoint`, which serves as a SOAP endpoint.

The SET clause specifies the name of the SOAP operation or service `FtoCService` that is to be called.

The default format used when making a web service request is 'SOAP:RPC'. The format chosen in this example is 'SOAP:DOC', which is similar to 'SOAP:RPC' but allows for a richer set of data types. SOAP requests are always sent as XML documents. The mechanism for sending SOAP requests is 'HTTP:POST'.

The substitution variables (`inoutheader`, `inheader`) in a web service client procedure like `FtoC` must be alpha-numeric. If the web service client is declared as a function, all its parameters are IN mode only (they cannot be assigned by the called function). Therefore, `OPENXML` or other string functions would have to be used to extract the SOAP response header information.

5. Create a wrapper procedure that builds two special SOAP request header entries, passes them to the `FtoC` procedure, and processes server responses.

Execute the following SQL statements in Interactive SQL:

```
CREATE OR REPLACE PROCEDURE FahrenheitToCelsius( Fahrenheit
    FLOAT )
    BEGIN
        DECLARE io_header LONG VARCHAR;
        DECLARE in_header LONG VARCHAR;
```

```

DECLARE result LONG VARCHAR;
DECLARE err INTEGER;
DECLARE crsr CURSOR FOR
    CALL FtoC( Fahrenheit io_header, in_header );
SET io_header =
    '<Authentication xmlns="SecretAgent" ' ||
    'mustUnderstand="1">' ||
    '<userName alias="99">' ||
    '<first>Susan</first><last>Hilton</last>' ||
    '</userName>' ||
    '</Authentication>';
SET in_header =
    '<Session xmlns="SomeSession">' ||
    '123456789' ||
    '</Session>';

MESSAGE 'send, soapheader=' || io_header || in_header;
OPEN crsr;
FETCH crsr INTO result, err;
CLOSE crsr;
MESSAGE 'receive, soapheader=' || io_header;
SELECT Fahrenheit, Celsius
    FROM OPENXML(result, '//tns:answer', 1, result)
    WITH ("Celsius" FLOAT 'text()');
END;

```

The first SET statement creates the XML representation of a SOAP header entry to inform the web server of user credentials:

```

<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>

```

The second SET statement creates the XML representation of a SOAP header entry to track the client session ID:

```

<Session xmlns="SomeSession">123456789</Session>

```

6. The OPEN statement causes the FtoC procedure to be called which sends a SOAP request to the web server and then processes the response from the web server. The response includes a header which is returned in inoutheader.

At this point, you now have a client that can send SOAP requests to the web server and receive SOAP responses from the web server.

Lesson 3: Sending a SOAP Request and Receiving a SOAP Response

In this lesson, you call the wrapper procedure created in the previous lesson, which sends a SOAP request to the web server that you created in lesson one.

Prerequisites

This lesson assumes that you have set up a web server as instructed in lesson 1.

This lesson assumes that you have set up a web client as instructed in lesson 2.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using SAP Sybase IQ to access a SOAP/DISH service.

Task

1. Connect to the client database in Interactive SQL if it is not already open from lesson two.

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc_client"
```

2. Enable logging of SOAP requests and responses.

Execute the following SQL statements in Interactive SQL:

```
CALL sa_server_option('WebClientLogFile', 'soap.txt');
CALL sa_server_option('WebClientLogging', 'ON');
```

These calls allow you to examine the content of the SOAP request and response. The requests and responses are logged to a file called `soap.txt`.

3. Call the wrapper procedure to send a SOAP request and receive the SOAP response.

Execute the following SQL statement in Interactive SQL:

```
CALL FahrenheitToCelsius (212);
```

This call passes a Fahrenheit value of 212 to the `FahrenheitToCelsius` procedure, which passes the value along with two customized SOAP headers to the `FTOC` procedure. Both client-side procedures are created in the previous lesson.

The `FTOC` web service procedure sends the Fahrenheit value and the SOAP headers to the web server. The SOAP request contains the following.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" mustUnderstand="1">
      <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:FtoCService>
      <m:fahrenheit>212</m:fahrenheit>
    </m:FtoCService>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `FtoC` procedure then receives the response from the web server which includes a result set based on the Fahrenheit value. The SOAP response contains the following.

```
<SOAP-ENV:Envelope
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:tns='http://localhost:8082'>
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" alias="99"
mustUnderstand="1">
      <first>Susan</first>
      <last>Hilton</last>
    </Authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <tns:FtoCServiceResponse>
      <tns:FtoCServiceResult xsi:type='xsd:string'>
        &lt;t;tns:rowset xmlns:tns="http://localhost:8082/
ftc" &gt; &#x0A;
          &lt;t;tns:row &gt; &#x0A;
            &lt;t;tns:answer &gt; 100
            &lt;/tns:answer &gt; &#x0A;
          &lt;/tns:row &gt; &#x0A;
          &lt;/tns:rowset &gt; &#x0A;
        </tns:FtoCServiceResult>
        <tns:sqlcode>0</tns:sqlcode>
      </tns:FtoCServiceResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The content of `<SOAP-ENV:Header>` is returned in `inoutheader`.

If you examine the SOAP response, you can see that the result set was encoded in the response by the `FtoCService` web service. The result set is decoded and returned to the `FahrenheitToCelsius` procedure. The result set looks like the following when a Fahrenheit value of 212 is passed to the web server:

```
<tns:rowset xmlns:tns="http://localhost:8082/ftc">
  <tns:row>
    <tns:answer>100
  </tns:row>
</tns:rowset>
```

The `SELECT` statement in the `FahrenheitToCelsius` procedure uses the `OPENXML` function to parse the SOAP response, extracting the Celsius value defined by the `tns:answer` structure.

The following result set is generated in Interactive SQL:

Fahrenheit	Celsius
212	100

Tutorial: Using Visual C# to Access a SOAP/DISH Web Service

This tutorial illustrates how to create a Visual C# client application to access SOAP/DISH services on an SAP Sybase IQ web server.

Required Software

- SAP Sybase IQ
- Visual Studio

Competencies and Experience

- Familiarity with SOAP
- Familiarity with .NET framework
- Basic knowledge of SQL Anywhere web services

Goals

- Create and start a new SAP Sybase IQ web server database.
- Create a SOAP web service.
- Set up a procedure that returns the information contained in a SOAP request.
- Create a DISH web service that provides WSDL documents and acts as a proxy.
- Set up Visual C# on the client computer and import a WSDL document from the web server.
- Create a Java client application to retrieve information from the SOAP service using the WSDL document information.

Privileges

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses

In this lesson, you set up an SAP Sybase IQ web server running SOAP and DISH web services that handles Visual C# client application requests.

Prerequisites

A recent version of Visual Studio is required.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using Visual C# to access a SOAP/DISH web service.

Task

1. Start the SAP Sybase IQ demonstration database using the following command:

```
iqsrv16 -xs http(port=8082) iqdemo.db
```

This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

2. Connect to the database server in Interactive SQL using the following command:

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=demo"
```

3. Create a new SOAP service to accept incoming requests.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE "SASoapTest/EmployeeList"
    TYPE 'SOAP'
    DATATYPE ON
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA
    AS SELECT * FROM Employees;
```

This statement creates a new SOAP web service named `SASoapTest/EmployeeList` that generates a SOAP type as output. It selects all columns from the `Employees` table and returns the result set to the client. The service name is surrounded by quotation marks because of the slash character (/) that appears in the service name.

`DATATYPE ON` indicates that explicit data type information is generated in the XML result set response and the input parameters. This option does not affect the WSDL document that is generated.

The `FORMAT` clause is not specified so the SOAP service format is dictated by the associated DISH service format which is declared in the next step.

4. Create a new DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE SASoapTest_DNET
    TYPE 'DISH'
    GROUP SASoapTest
    FORMAT 'DNET'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA;
```

DISH web services accessed from .NET should be declared with the `FORMAT 'DNET'` clause. The `GROUP` clause identifies the SOAP services that should be handled by the DISH service. The `EmployeeList` service created in the previous step is part of the `GROUP SASoapTest` because it is declared as `SASoapTest/EmployeeList`.

5. Verify that the DISH web service is functional by accessing the associated WSDL document through a web browser.

Open your web browser and go to *http://localhost:8082/demo/SASoapTest_DNET*.

The DISH service automatically generates a WSDL document that appears in the browser window.

You have set up an SAP Sybase IQ web server running SOAP and DISH web services that can handle Visual C# client application requests.

Next

In the next lesson, you create a Visual C# application to communicate with the web server.

Lesson 2: Creating a Visual C# Application to Communicate with the Web Server

In this lesson, you create a Visual C# application to communicate with the web server.

Prerequisites

This lesson assumes that you have set up a web server as instructed in lesson 1.

A recent version of Visual Studio is required to complete this lesson.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using Visual C# to access a SOAP/DISH web service.

Task

This lesson contains several references to `localhost`. Use the host name or IP address of the web server from lesson 1 instead of `localhost` if you are not running the web client on the same computer as the web server.

This example uses functions from the .NET Framework 2.0.

1. Start Visual Studio.
2. Create a new Visual C# **Windows Forms Application** project.

An empty form appears.

3. Add a web reference to the project.
 - a. Click **Project » Add Service Reference**.
 - b. In the Add Service Reference window, click **Advanced**.
 - c. In the Service Reference Settings window, click **Add Web Reference**.
 - d. In the Add Web Reference window, type `http://localhost:8082/demo/SASoapTest_DNET` in the **URL** field.
 - e. Click **Go** (or the green arrow).

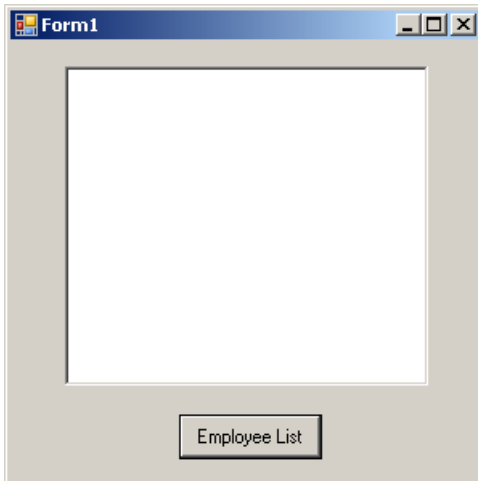
Visual Studio lists the `EmployeeList` method available from the `SASoapTest_DNET` service.

f. Click Add Reference.

Visual Studio adds `localhost` to the project **Web References** in the **Solution Explorer** pane.

4. Populate the empty form with the desired objects for web client application.

From the **Toolbox** pane, drag `ListBox` and `Button` objects onto the form and update the text attributes so that the form looks similar to the following diagram:



5. Write a procedure that accesses the web reference and uses the available methods.

Double-click the `Employee List` button and add the following code for the button click event:

```
int sqlCode;

listBox1.Items.Clear();

localhost.SASoapTest_DNET proxy = new
localhost.SASoapTest_DNET();

DataSet results = proxy.EmployeeList(out sqlCode);
DataTableReader dr = results.CreateDataReader();
while (dr.Read())
{
    for (int i = 0; i < dr.FieldCount; i++)
    {
        string columnName = "(" + dr.GetDataTypeName(i)
            + ")"
            + dr.GetName(i);
        if (dr.IsDBNull(i))
        {
            listBox1.Items.Add(columnName + "=(null)");
        }
        else {
            System.TypeCode typeCode =
```

```

        System.Type.GetTypeCode(dr.GetFieldType(i));
switch (typeCode)
{
    case System.TypeCode.Int32:
        Int32 intValue = dr.GetInt32(i);
        listBox1.Items.Add(columnName + "="
            + intValue);
        break;
    case System.TypeCode.Decimal:
        Decimal decValue = dr.GetDecimal(i);
        listBox1.Items.Add(columnName + "="
            + decValue.ToString("c"));
        break;
    case System.TypeCode.String:
        string stringValue = dr.GetString(i);
        listBox1.Items.Add(columnName + "="
            + stringValue);
        break;
    case System.TypeCode.DateTime:
        DateTime dateValue = dr.GetDateTime(i);
        listBox1.Items.Add(columnName + "="
            + dateValue);
        break;
    case System.TypeCode.Boolean:
        Boolean boolValue = dr.GetBoolean(i);
        listBox1.Items.Add(columnName + "="
            + boolValue);
        break;
    case System.TypeCode.DBNull:
        listBox1.Items.Add(columnName
            + "=(null)");
        break;
    default:
        listBox1.Items.Add(columnName
            + "=(unsupported)");
        break;
}
}
listBox1.Items.Add("");
}
dr.Close();

```

6. Run the application.

Click **Debug** » **Start Debugging**.

7. Communicate with the web database server.

Click **Employee List**.

The `ListBox` object displays the `EmployeeList` result set as (type)name=value pairs. The following output illustrates how an entry appears in the `ListBox` object:

```

(Int32) EmployeeID=102
(Int32) ManagerID=501
(String) Surname=Whitney

```

```
(String) GivenName=Fran
(Int32) DepartmentID=100
(String) Street=9 East Washington Street
(String) City=Cornwall
(String) State=New York
(String) Country=USA
(String) PostalCode=02192
(String) Phone=6175553985
(String) Status=A
(String) SocialSecurityNumber=017349033
(String) Salary=$45,700.00
(DateTime) StartDate=28/08/1984 0:00:00 AM
(DateTime) TerminationDate=(null)
(DateTime) BirthDate=05/06/1958 0:00:00 AM
(Boolean) BenefitHealthInsurance=True
(Boolean) BenefitLifeInsurance=True
(Boolean) BenefitDayCare=False
(String) Sex=F
```

The Salary amount is converted to the client's currency format.

Values that contain null are returned as DBNull. Values that contain a date with no time are assigned a time of 00:00:00 or midnight.

The XML response from the web server includes a formatted result set. All column data is converted to a string representation of the data. The following result set illustrates how result sets are formatted when they are sent to the client:

```
<row>
  <EmployeeID>102</EmployeeID>
  <ManagerID>501</ManagerID>
  <Surname>Whitney</Surname>
  <GivenName>Fran</GivenName>
  <DepartmentID>100</DepartmentID>
  <Street>9 East Washington Street</Street>
  <City>Cornwall</City>
  <State>NY</State>
  <Country>USA</Country>
  <PostalCode>02192</PostalCode>
  <Phone>6175553985</Phone>
  <Status>A</Status>
  <SocialSecurityNumber>017349033</SocialSecurityNumber>
  <Salary>45700.000</Salary>
  <StartDate>1984-08-28-05:00</StartDate>
  <TerminationDate xsi:nil="true" />
  <BirthDate>1958-06-05-05:00</BirthDate>
  <BenefitHealthInsurance>1</BenefitHealthInsurance>
  <BenefitLifeInsurance>1</BenefitLifeInsurance>
  <BenefitDayCare>0</BenefitDayCare>
  <Sex>F</Sex>
</row>
```

Columns containing date or time information include the offset from UTC of the web server. In the above result set, the offset is -05:00 which is 5 hours to the west of UTC (North American Eastern Standard Time).

Columns containing only the date are formatted as yyyy-mm-dd-HH:MM or yyyy-mm-dd+HH:MM. A zone offset (-HH:MM or +HH:MM) is suffixed to the string.

Columns containing only the time are formatted as hh:mm:ss.nnn-HH:MM or hh:mm:ss.nnn+HH:MM. A zone offset (-HH:MM or +HH:MM) is suffixed to the string.

Columns containing both date and time are formatted as yyyy-mm-ddThh:mm:ss.nnn-HH:MM or yyyy-mm-ddThh:mm:ss.nnn+HH:MM. The date is separated from the time using the letter 'T'. A zone offset (-HH:MM or +HH:MM) is suffixed to the string.

The DATATYPE ON clause was specified in the previous lesson to generate data type information in the XML result set response. A fragment of the response from the web server is shown below. The type information matches the data type of the database columns.

```
<xsd:element name='EmployeeID' minOccurs='0' type='xsd:int' />
<xsd:element name='ManagerID' minOccurs='0' type='xsd:int' />
<xsd:element name='Surname' minOccurs='0' type='xsd:string' />
<xsd:element name='GivenName' minOccurs='0' type='xsd:string' />
<xsd:element name='DepartmentID' minOccurs='0' type='xsd:int' />
<xsd:element name='Street' minOccurs='0' type='xsd:string' />
<xsd:element name='City' minOccurs='0' type='xsd:string' />
<xsd:element name='State' minOccurs='0' type='xsd:string' />
<xsd:element name='Country' minOccurs='0' type='xsd:string' />
<xsd:element name='PostalCode' minOccurs='0' type='xsd:string' />
<xsd:element name='Phone' minOccurs='0' type='xsd:string' />
<xsd:element name='Status' minOccurs='0' type='xsd:string' />
<xsd:element name='SocialSecurityNumber' minOccurs='0'
type='xsd:string' />
<xsd:element name='Salary' minOccurs='0' type='xsd:decimal' />
<xsd:element name='StartDate' minOccurs='0' type='xsd:date' />
<xsd:element name='TerminationDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BirthDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BenefitHealthInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitLifeInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitDayCare' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='Sex' minOccurs='0' type='xsd:string' />
```

Tutorial: Using JAX-WS to Access a SOAP/DISH Web Service

This tutorial illustrates how to create a Java API for XML Web Services (JAX-WS) client application to access SOAP/DISH services on a SQL Anywhere web server.

Required Software

- SAP Sybase IQ
- JDK 1.7.0
- JAX-WS 2.2.7 or later version

Competencies and Experience

- Familiarity with SOAP
- Familiarity with Java and JAX-WS
- Basic knowledge of SAP Sybase IQ web services

Goals

- Create and start a new SAP Sybase IQ web server database.
- Create a SOAP web service.
- Set up a procedure that returns the information contained in a SOAP request.
- Create a DISH web service that provides WSDL documents and acts as a proxy.
- Use JAX-WS on the client computer to process a WSDL document from the web server.
- Create a Java client application to retrieve information from the SOAP service using the WSDL document information.

Privileges

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses

In this lesson, you set up an SAP Sybase IQ web server running SOAP and DISH web services that handles JAX-WS client application requests.

Prerequisites

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using JAX-WS to access a SOAP/DISH web service.

Task

This lesson sets up the web server and a simple web service that you will use in the next lesson. It can be instructional to use proxy software to observe the XML message traffic. The proxy inserts itself between your client application and the web server.

1. Start the SAP Sybase IQ demonstration database using the following command:

```
iqsrv16 -xs http(port=8082) iqdemo.db
```

This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

2. Connect to the database server with Interactive SQL using the following command:

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=demo"
```

3. Create a stored procedure that lists Employees table columns.

Execute the following SQL statements in Interactive SQL:

```
CREATE OR REPLACE PROCEDURE ListEmployees()
RESULT (
  EmployeeID          INTEGER,
  Surname             CHAR(20),
  GivenName           CHAR(20),
  StartDate           DATE,
  TerminationDate     DATE )
BEGIN
  SELECT EmployeeID, Surname, GivenName, StartDate,
  TerminationDate
  FROM Employees;
END;
```

These statements create a new procedure named `ListEmployees` that defines the structure of the result set output, and selects certain columns from the `Employees` table.

4. Create a new SOAP service to accept incoming requests.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE "WS/EmployeeList"
  TYPE 'SOAP'
  FORMAT 'CONCRETE' EXPLICIT ON
  DATATYPE ON
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA
  AS CALL ListEmployees();
```

This statement creates a new SOAP web service named `WS/EmployeeList` that generates a SOAP type as output. It calls the `ListEmployees` procedure when a web client sends a request to the service. The service name is surrounded by quotation marks because of the slash character (/) that appears in the service name.

SOAP web services accessed from JAX-WS should be declared with the `FORMAT 'CONCRETE'` clause. The `EXPLICIT ON` clause indicates that the corresponding DISH service should generate XML Schema that describes an explicit dataset object based on the result set of the `ListEmployees` procedure. The `EXPLICIT` clause only affects the generated WSDL document.

`DATATYPE ON` indicates that explicit data type information is generated in the XML result set response and the input parameters. This option does not affect the WSDL document that is generated.

5. Create a new DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE WSDish
  TYPE 'DISH'
  FORMAT 'CONCRETE'
  GROUP WS
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA;
```

DISH web services accessed from JAX-WS should be declared with the `FORMAT 'CONCRETE'` clause. The `GROUP` clause identifies the SOAP services that should be handled by the DISH service. The `EmployeeList` service created in the previous step is part of the `GROUP WS` because it is declared as `WS/EmployeeList`.

6. Verify that the DISH web service is functional by accessing the associated WSDL document through a web browser.

Open your web browser and go to <http://localhost:8082/demo/WSDish>.

The DISH service automatically generates a WSDL document that appears in the browser window. Examine the `EmployeeListDataset` object, which looks similar to the following output:

```
<s:complexType name="EmployeeListDataset">
  <s:sequence>
    <s:element name="rowset">
      <s:complexType>
        <s:sequence>
          <s:element name="row" minOccurs="0" maxOccurs="unbounded">
            <s:complexType>
              <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="EmployeeID"
nillable="true" type="s:int" />
                <s:element minOccurs="0" maxOccurs="1" name="Surname"
nillable="true" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="GivenName"
nillable="true" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="StartDate"
nillable="true" type="s:date" />
                <s:element minOccurs="0" maxOccurs="1" name="TerminationDate"
nillable="true" type="s:date" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:sequence>
</s:complexType>
```

`EmployeeListDataset` is the explicit object generated by the `FORMAT 'CONCRETE'` and `EXPLICIT ON` clauses in the `EmployeeList` SOAP service. In a later lesson, the `wimport` application uses this information to generate a SOAP 1.1 client interface for this service.

You have set up an SAP Sybase IQ web server running SOAP and DISH web services that can handle JAX-WS client application requests.

Next

In the next lesson, you create a Java application to communicate with the web server.

Lesson 2: Creating a Java Application to Communicate with the Web Server

In this lesson, you process the WSDL document generated from the DISH service and create a Java application to access table data based on the schema defined in the WSDL document.

Prerequisites

This lesson depends on the steps carried out in lesson 1.

This lesson assumes that you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Using JAX-WS to access a SOAP/DISH web service.

Task

At the time of writing, JAX-WS is included in JDK 1.7.0 and the most recent version of JAX-WS was 2.2.7. The steps that follow are based on that version. To determine if JAX-WS is present in your JDK, check for the `wsimport` application in the JDK `bin` directory. If it is not there then go to <http://jax-ws.java.net/> to download and install the latest version of JAX-WS.

This lesson contains several references to `localhost`. Use the host name or IP address of the web server from lesson 1 instead of `localhost` if you are not running the web client on the same computer as the web server.

1. At a command prompt, create a new working directory for your Java code and generated files. Change to this new directory.
2. Generate the interface that calls the DISH web service and imports the WSDL document using the following command:

```
wsimport -keep "http://localhost:8082/demo/WSdish"
```

The `wsimport` application retrieves the WSDL document from the given URL. It generates `.java` files to create an interface for it, then compiles them into `.class` files.

The `keep` option indicates that the `.java` files should not be deleted after generating the class files. The generated Java source code allows you to understand the generated class files.

The `wsimport` application creates a new subdirectory structure named `localhost_8082\demo\ws` in your current working directory. The following is a list of the contents of directory `ws`:

- `EmployeeList.class`
- `EmployeeList.java`
- `EmployeeListDataset$Rowset$Row.class`
- `EmployeeListDataset$Rowset.class`
- `EmployeeListDataset.class`
- `EmployeeListDataset.java`
- `EmployeeListResponse.class`
- `EmployeeListResponse.java`
- `FaultMessage.class`
- `FaultMessage.java`
- `ObjectFactory.class`
- `ObjectFactory.java`
- `package-info.class`
- `package-info.java`
- `WSDish.class`
- `WSDish.java`
- `WSDishSoapPort.class`
- `WSDishSoapPort.java`

3. Write a Java application that accesses table data from the database server based on the dataset object schema defined in the generated source code.

The following is a sample Java application that does this. Save the source code as `SASoapDemo.java` in the current working directory. Your current working directory must be the directory containing the `localhost` subdirectory.

```
// SASoapDemo.java illustrates a web service client that  
// calls the WSDish service and prints out the data.
```

```
import java.util.*;  
import javax.xml.ws.*;  
import org.w3c.dom.Element;  
import org.w3c.dom.Node;  
import javax.xml.datatype.*;  
import localhost._8082.demo.ws.*;  
  
public class SASoapDemo  
{  
    public static void main( String[] args )  
    {  
        try {  
            WSDish service = new WSDish();  
  
            Holder<EmployeeListDataset> response =  
                new Holder<EmployeeListDataset>();  
            Holder<Integer> sqlcode = new Holder<Integer>();  
  
            WSDishSoapPort port = service.getWSDishSoap();
```

```

// This is the SOAP service call to EmployeeList
port.employeeList( response, sqlcode );

EmployeeListDataset result = response.value;
EmployeeListDataset.Rowset rowset = result.getRowset();

List<EmployeeListDataset.Rowset.Row> rows = rowset.getRow();

String fieldType;
String fieldName;
String fieldValue;
Integer fieldInt;
XMLGregorianCalendar fieldDate;

for ( int i = 0; i < rows.size(); i++ ) {
    EmployeeListDataset.Rowset.Row row = rows.get( i );

    fieldType =
row.getEmployeeID().getDeclaredType().getSimpleName();
    fieldName = row.getEmployeeID().getName().getLocalPart();
    fieldInt = row.getEmployeeID().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
                        "=" + fieldInt );

    fieldType =
row.getSurname().getDeclaredType().getSimpleName();
    fieldName = row.getSurname().getName().getLocalPart();
    fieldValue = row.getSurname().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
                        "=" + fieldValue );

    fieldType =
row.getGivenName().getDeclaredType().getSimpleName();
    fieldName = row.getGivenName().getName().getLocalPart();
    fieldValue = row.getGivenName().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
                        "=" + fieldValue );

    fieldType =
row.getStartDate().getDeclaredType().getSimpleName();
    fieldName = row.getStartDate().getName().getLocalPart();
    fieldDate = row.getStartDate().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
                        "=" + fieldDate );

    if ( row.getTerminationDate() == null ) {
        fieldType = "unknown";
        fieldName = "TerminationDate";
        fieldDate = null;
    } else {
        fieldType =
row.getTerminationDate().getDeclaredType().getSimpleName();
        fieldName =
row.getTerminationDate().getName().getLocalPart();
        fieldDate = row.getTerminationDate().getValue();
    }
}

```

```
    }
    System.out.println( "(" + fieldType + ")" + fieldName +
                        "=" + fieldDate );
    System.out.println();
  }
}
catch (Exception x) {
    x.printStackTrace();
}
}
```

This application prints all server-provided column data to the standard system output.

Note: This application assumes that your SAP Sybase IQ web server is listening on port 8082, as instructed in lesson one. Replace the 8082 portion of the `import localhost._8082.demo.ws.*` code line with the port number you specified when you started the SAP Sybase IQ web server.

For more information about the Java methods used in this application, see the `javax.xml.bind.JAXBElement` class API documentation at <http://docs.oracle.com/javase/>.

4. Compile your Java application using the following command:

```
javac SASoapDemo.java
```

5. Execute the application using the following command:

```
java SASoapDemo
```

6. The application sends its request to the web server. It receives an XML result set response that consists of an `EmployeeListResult` with a rowset containing several row entries.

The following is an example of the output from running `SASoapDemo`:

```
(Integer) EmployeeID=102
(String) Surname=Whitney
(String) GivenName=Fran
(XMLGregorianCalendar) StartDate=1984-08-28
(unknown) TerminationDate=null

(Integer) EmployeeID=105
(String) Surname=Cobb
(String) GivenName=Matthew
(XMLGregorianCalendar) StartDate=1985-01-01
(unknown) TerminationDate=null
.
.
.
(Integer) EmployeeID=1740
(String) Surname=Nielsen
(String) GivenName=Robert
(XMLGregorianCalendar) StartDate=1994-06-24
(unknown) TerminationDate=null
```

```
(Integer) EmployeeID=1751
(String) Surname=Ahmed
(String) GivenName=Alex
(XMLGregorianCalendar) StartDate=1994-07-12
(XMLGregorianCalendar) TerminationDate=2008-04-18
```

The `TerminationDate` column is only sent when its value is not NULL. The Java application is designed to detect when the `TerminationDate` column is not present. For this example, the last row in the `Employees` table was altered such that a non-NULL termination date was set.

The following is an example of a SOAP response from the web server. The `SQLCODE` result from executing the query is included in the response.

```
<tns:EmployeeListResponse>
  <tns:EmployeeListResult xsi:type='tns:EmployeeListDataset'>
    <tns:rowset>
      <tns:row> ... </tns:row>
      .
      .
      .
      <tns:row>
        <tns:EmployeeID xsi:type="xsd:int">1751</tns:EmployeeID>
        <tns:Surname xsi:type="xsd:string">Ahmed</tns:Surname>
        <tns:GivenName xsi:type="xsd:string">Alex</tns:GivenName>
        <tns:StartDate xsi:type="xsd:dateTime">1994-07-12</
tns:StartDate>
        <tns:TerminationDate xsi:type="xsd:dateTime">2010-03-22</
tns:TerminationDate>
      </tns:row>
    </tns:rowset>
  </tns:EmployeeListResult>
  <tns:sqlcode>0</tns:sqlcode>
</tns:EmployeeListResponse>
```

Column names and data types are included in each rowset.

Three-Tier Computing and Distributed Transactions

You can use SAP Sybase IQ as a database server or *resource manager*, participating in distributed transactions coordinated by a transaction server.

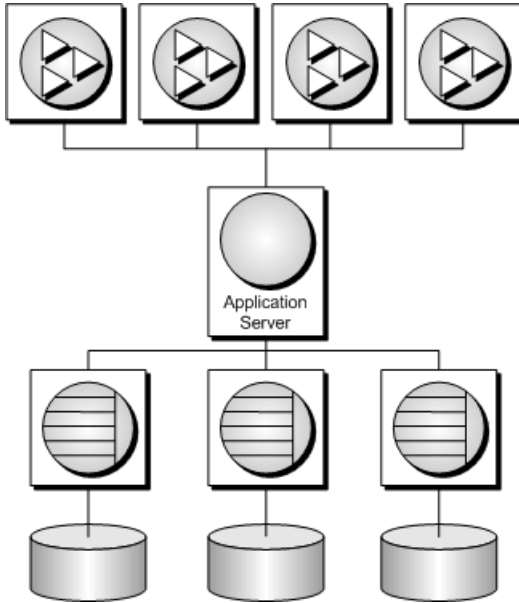
A three-tier environment, where an application server sits between client applications and a set of resource managers, is a common distributed-transaction environment. Sybase EAServer and some other application servers are also transaction servers.

Sybase EAServer and Microsoft Transaction Server both use the Microsoft Distributed Transaction Coordinator (DTC) to coordinate transactions. SAP Sybase IQ provides support for distributed transactions controlled by the DTC service, so you can use SAP Sybase IQ with either of these application servers, or any other product based on the DTC model.

When integrating SAP Sybase IQ into a three-tier environment, most of the work needs to be done from the application server. This section provides an introduction to the concepts and architecture of three-tier computing, and an overview of relevant SAP Sybase IQ features. It does not describe how to configure your application server to work with SAP Sybase IQ. For more information, see your application server documentation.

Three-Tier Computing Architecture

In three-tier computing, application logic is held in an application server, such as Sybase EAServer, which sits between the resource manager and the client applications. In many situations, a single application server may access multiple resource managers. In the Internet case, client applications are browser-based, and the application server is generally a web server extension.



Sybase EAServer stores application logic in the form of components, and makes these components available to client applications. The components may be PowerBuilder® components, JavaBeans, or COM components.

For more information, see your Sybase EAServer documentation.

Distributed Transactions in Three-Tier Computing

When client applications or application servers work with a single transaction processing database, such as SAP Sybase IQ, there is no need for transaction logic outside the database itself, but when working with multiple resource managers, transaction control must span the resources involved in the transaction. Application servers provide transaction logic to their client applications—guaranteeing that sets of operations are executed atomically.

Many transaction servers, including Sybase EAServer, use the Microsoft Distributed Transaction Coordinator (DTC) to provide transaction services to their client applications. DTC uses *OLE transactions*, which in turn use the *two-phase commit* protocol to coordinate transactions involving multiple resource managers. You must have DTC installed to use the features described in this section.

SAP Sybase IQ in Distributed Transactions

SAP Sybase IQ can take part in transactions coordinated by DTC, which means that you can use SAP Sybase IQ databases in distributed transactions using a transaction server such as Sybase EAServer or Microsoft Transaction Server. You can also use DTC directly in your applications to coordinate transactions across multiple resource managers.

The Vocabulary of Distributed Transactions

This section assumes some familiarity with distributed transactions. For information, see your transaction server documentation. This section describes some commonly used terms.

- *Resource managers* are those services that manage the data involved in the transaction. The SAP Sybase IQ database server can act as a resource manager in a distributed transaction when accessed through ADO.NET, OLE DB, or ODBC. The SAP Sybase IQ .NET Data Provider, OLE DB provider, and ODBC driver act as resource manager proxies on the client computer. The SAP Sybase IQ .NET Data Provider supports distributed transactions using DbProviderFactory and TransactionScope.
- Instead of communicating directly with the resource manager, application components can communicate with *resource dispensers*, which in turn manage connections or pools of connections to the resource managers.
SAP Sybase IQ supports two resource dispensers: the ODBC driver manager and OLE DB.
- When a transactional component requests a database connection (using a resource manager), the application server *enlists* each database connection that takes part in the transaction. DTC and the resource dispenser perform the enlistment process.

Two-Phase Commit

Distributed transactions are managed using two-phase commit. When the work of the transaction is complete, the transaction manager (DTC) asks all the resource managers enlisted in the transaction whether they are ready to commit the transaction. This phase is called *preparing* to commit.

If all the resource managers respond that they are prepared to commit, DTC sends a commit request to each resource manager, and responds to its client that the transaction is completed. If one or more resource manager does not respond, or responds that it cannot commit the transaction, all the work of the transaction is rolled back across all resource managers.

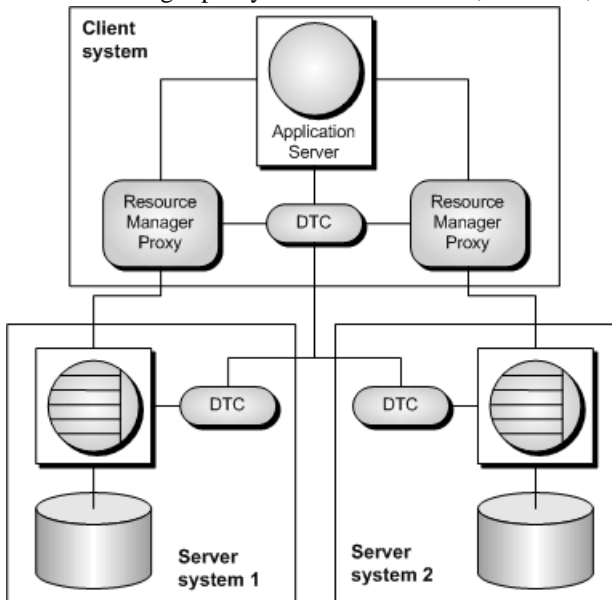
How Application Servers Use DTC

Sybase EAServer and Microsoft Transaction Server are both component servers. The application logic is held in the form of components, and made available to client applications.

Each component has a transaction attribute that indicates how the component participates in transactions. When building the component, you must program the work of the transaction into the component—the resource manager connections, the operations on the data for which each resource manager is responsible. However, you do not need to add transaction management logic to the component. Once the transaction attribute is set, to indicate that the component needs transaction management, EAServer uses DTC to enlist the transaction and manage the two-phase commit process.

Distributed Transaction Architecture

The following diagram illustrates the architecture of distributed transactions. In this case, the resource manager proxy is either ADO.NET, OLE DB, or ODBC.



In this case, a single resource dispenser is used. The application server asks DTC to prepare a transaction. DTC and the resource dispenser enlist each connection in the transaction. Each resource manager must be in contact with both the DTC and the database, so the work can be performed and the DTC can be notified of its transaction status when required.

A Distributed Transaction Coordinator (DTC) service must be running on each computer to operate distributed transactions. You can start or stop DTC from the Microsoft Windows **Services** window; the DTC service task is named MSDTC.

For more information, see your DTC or EA Server documentation.

Distributed Transactions

While SAP Sybase IQ is enlisted in a distributed transaction, it hands transaction control over to the transaction server, and SAP Sybase IQ ensures that it does not perform any implicit transaction management. The following conditions are imposed automatically by SAP Sybase IQ when it participates in distributed transactions:

- Autocommit is automatically turned off, if it is in use.
- Data definition statements (which commit as a side effect) are disallowed during distributed transactions.

- An explicit COMMIT or ROLLBACK issued by the application directly to SAP Sybase IQ, instead of through the transaction coordinator, generates an error. The transaction is not aborted, however.
- A connection can participate in only a single distributed transaction at a time.
- There must be no uncommitted operations at the time the connection is enlisted in a distributed transaction.

DTC Isolation Levels

DTC has a set of isolation levels, which the application server specifies. These isolation levels map to SAP Sybase IQ isolation levels as follows:

DTC isolation level	SAP Sybase IQ isolation level
ISOLATIONLEVEL_UNSPECIFIED	0
ISOLATIONLEVEL_CHAOS	0
ISOLATIONLEVEL_READUNCOMMITTED	0
ISOLATIONLEVEL_BROWSE	0
ISOLATIONLEVEL_CURSORSTABILITY	1
ISOLATIONLEVEL_READCOMMITTED	1
ISOLATIONLEVEL_REPEATABLEREAD	2
ISOLATIONLEVEL_SERIALIZABLE	3
ISOLATIONLEVEL_ISOLATED	3

Recovery From Distributed Transactions

If the database server faults while uncommitted operations are pending, it must either rollback or commit those operations on startup to preserve the atomic nature of the transaction.

If uncommitted operations from a distributed transaction are found during recovery, the database server attempts to connect to DTC and requests that it be re-enlisted in the pending or in-doubt transactions. Once the re-enlistment is complete, DTC instructs the database server to roll back or commit the outstanding operations.

If the reenlistment process fails, SAP Sybase IQ has no way of knowing whether the in-doubt operations should be committed or rolled back, and recovery fails. If you want the database in such a state to recover, regardless of the uncertain state of the data, you can force recovery using the following database server options:

- **-tmf** – If DTC cannot be located, the outstanding operations are rolled back and recovery continues.
- **-tmt** – If re-enlistment is not achieved before the specified time, the outstanding operations are rolled back and recovery continues.

Database Tools Interface (DBTools)

SAP Sybase IQ includes Sybase Control Center and a set of utilities for managing databases. These database management utilities perform tasks such as backing up databases, creating databases, translating transaction logs to SQL, and so on.

Supported Platforms

All the database management utilities use a shared library called the *database tools library*. It is supplied for Windows operating systems and for Linux, and Unix. For Windows, the name of this library is `dbtool116.dll`. For Linux and Unix, the name of this library is `libdbtool116_r.so`.

You can develop your own database management utilities or incorporate database management features into your applications by calling the database tools library. This section describes the interface to the database tools library. This section assumes you are familiar with how to call library routines from the development environment you are using.

The database tools library has functions, or entry points, for each of the database management utilities. In addition, functions must be called before use of other database tools functions and when you have finished using other database tools functions.

The dbtools.h Header File

The `dbtools.h` header file lists the entry points to the DBTools library and also the structures used to pass information to and from the library. The `dbtools.h` file is installed into the `SDK\Include` subdirectory under your SAP Sybase IQ installation directory. You should consult the `dbtools.h` file for the latest information about the entry points and structure members.

The `dbtools.h` header file includes other files such as:

- **`sqlca.h`** – This is included for resolution of various macros, not for the SQLCA itself.
- **`dllapi.h`** – Defines preprocessor macros for operating-system dependent and language-dependent macros.
- **`dbtlvers.h`** – Defines the `DB_TOOLS_VERSION_NUMBER` preprocessor macro and other version specific macros.

The sqldef.h Header File

The `sqldef.h` header file includes error return values.

The dbrmt.h Header File

The `dbrmt.h` header file included with SAP Sybase IQ describes the DBRemoteSQL entry point in the DBTools library and also the structure used to pass information to and from the DBRemoteSQL entry point. The `dbrmt.h` file is installed into the `SDK\Include`

Database Tools Interface (DBTools)

subdirectory under your SAP Sybase IQ installation directory. You should consult the `dbtmt.h` file for the latest information about the `DBRemoteSQL` entry point and structure members.

DBTools Import Libraries

To use the DBTools functions, you must link your application against a DBTools *import library* that contains the required function definitions.

For Unix systems, no import library is required. Link directly against `libdbtool16.so` (non-threaded) or `libdbtool16_r.so` (threaded).

Import libraries

Import libraries for the DBTools interface are provided with SAP Sybase IQ for Windows. For Windows, they can be found in the `SDK\Lib\x86` and `SDK\Lib\x64` subdirectories under your SAP Sybase IQ installation directory. The provided DBTools import libraries are as follows:

Compiler	Library
Microsoft Windows	<code>dbtlstm.lib</code>

DBTools Library Initialization and Finalization

Before using any other DBTools functions, you must call `DBToolsInit`. When you are finished using the DBTools library, you must call `DBToolsFini`.

The primary purpose of the `DBToolsInit` and `DBToolsFini` functions is to allow the DBTools library to load and unload the SAP Sybase IQ message library. The message library contains localized versions of all error messages and prompts that DBTools uses internally. If `DBToolsFini` is not called, the reference count of the messages library is not decremented and it will not be unloaded, so be careful to ensure there is a matched pair of `DBToolsInit/DBToolsFini` calls.

The following code fragment illustrates how to initialize and finalize DBTools:

```
// Declarations
a_dbtools_info  info;
short          ret;

//Initialize the a_dbtools_info structure
memset( &info, 0, sizeof( a_dbtools_info ) );
info.errorrtn = (MSG_CALLBACK)MyErrorRtn;

// initialize the DBTools library
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
```

```

    // library initialization failed
    ...
}
// call some DBTools routines ...
...
// finalize the DBTools library
DBToolsFini( &info );

```

DBTools Function Calls

All the tools are run by first filling out a structure, and then calling a function (or *entry point*) in the DBTools library. Each entry point takes a pointer to a single structure as argument.

The following example shows how to use the DBBackup function on a Windows operating system.

```

// Initialize the structure
a_backup_db backup_info;
memset( &backup_info, 0, sizeof( backup_info ) );

// Fill out the structure
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.output_dir = "c:\\backup";
backup_info.connectparms
="UID=<user_id>;PWD=<password>;DBF=iqdemo.db";

backup_info.confirmrtn = (MSG_CALLBACK) ConfirmRtn ;
backup_info.errorrtn = (MSG_CALLBACK) ErrorRtn ;
backup_info.msgrtn = (MSG_CALLBACK) MessageRtn ;
backup_info.statusrtn = (MSG_CALLBACK) StatusRtn ;
backup_info.backup_database = TRUE;

// start the backup
DBBackup( &backup_info );

```

Callback Functions

Several elements in DBTools structures are of type MSG_CALLBACK. These are pointers to callback functions.

Uses of Callback Functions

Callback functions allow DBTools functions to return control of operation to the user's calling application. The DBTools library uses callback functions to handle messages sent to the user by the DBTools functions for four purposes:

- **Confirmation** – Called when an action needs to be confirmed by the user. For example, if the backup directory does not exist, the tools library asks if it needs to be created.
- **Error message** – Called to handle a message when an error occurs, such as when an operation is out of disk space.

- **Information message** – Called for the tools to display some message to the user (such as the name of the current table being unloaded).
- **Status information** – Called for the tools to display the status of an operation (such as the percentage done when unloading a table).

Assigning a Callback Function to a Structure

You can directly assign a callback routine to the structure. The following statement is an example using a backup structure:

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

MSG_CALLBACK is defined in the `dllapi.h` header file supplied with SAP Sybase IQ. Tools routines can call back to the calling application with messages that should appear in the appropriate user interface, whether that be a windowing environment, standard output on a character-based system, or other user interface.

Confirmation Callback Function Example

The following example confirmation routine asks the user to answer YES or NO to a prompt and returns the user's selection:

```
extern short _callback ConfirmRtn(
    char * question )
{
    int ret = IDNO;
    if( question != NULL ) {
        ret = MessageBox( HwndParent, question,
            "Confirm", MB_ICONEXCLAMATION|MB_YESNO );
    }
    return( ret == IDYES );
}
```

Error Callback Function Example

The following is an example of an error message handling routine, which displays the error message in a window.

```
extern short _callback ErrorRtn(
    char * errorstr )
{
    if( errorstr != NULL ) {
        MessageBox( HwndParent, errorstr, "Backup Error",
MB_ICONSTOP|MB_OK );
    }
    return( 0 );
}
```

Message callback function example

A common implementation of a message callback function outputs the message to the screen:

```
extern short _callback MessageRtn(
    char * messagestr )
{
    if( messagestr != NULL ) {
```



```

        OutputMessageToWindow( messagestr );
    }
    return( 0 );
}

```

Status Callback Function Example

A status callback routine is called when a tool needs to display the status of an operation (like the percentage done unloading a table). A common implementation would just output the message to the screen:

```

extern short _callback StatusRtn(
    char * statusstr )
{
    if( statusstr != NULL ) {
        OutputMessageToWindow( statusstr );
    }
    return( 0 );
}

```

Version Numbers and Compatibility

Each structure has a member that indicates the version number. You should set the version field to the version number of the DBTools library that your application was developed against before calling any DBTools function. The current version of the DBTools library is defined when you include the `dbtools.h` header file.

The following example assigns the current version to an instance of the `a_backup_db` structure:

```

backup_info.version = DB_TOOLS_VERSION_NUMBER;

```

The version number allows your application to continue working with newer versions of the DBTools library. The DBTools functions use the version number supplied by your application to allow the application to work, even if new members have been added to the DBTools structure.

When any of the DBTools structures are updated, or when a newer version of the software is released, the version number is augmented. If you use `DB_TOOLS_VERSION_NUMBER` and you rebuild your application with a new version of the DBTools header file, then you must deploy a new version of the DBTools library.

Bit Fields

Many of the DBTools structures use bit fields to hold Boolean information in a compact manner. For example, the backup structure includes the following bit fields:

```

a_bit_field    backup_database : 1;
a_bit_field    backup_logfile  : 1;
a_bit_field    no_confirm     : 1;
a_bit_field    quiet          : 1;

```

Database Tools Interface (DBTools)

```
a_bit_field    rename_log    : 1;
a_bit_field    truncate_log  : 1;
a_bit_field    rename_local_log: 1;
a_bit_field    server_backup  : 1;
```

Each bit field is one bit long, indicated by the 1 to the right of the colon in the structure declaration. The specific data type used depends on the value assigned to `a_bit_field`, which is set at the top of `dbtools.h`, and is operating system-dependent.

You assign a value of 0 or 1 to a bit field to pass Boolean information in the structure.

A DBTools Example

You can find this sample and instructions for compiling it in the `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\DBTools` directory. The sample program itself is in `main.cpp`. The sample illustrates how to use the DBTools library to perform a backup of a database.

```
#define WIN32

#include <stdio.h>
#include <string.h>
#include "windows.h"
#include "sqldef.h"
#include "dbtools.h"
extern short _callback ConfirmCallBack( char * str )
{
    if( MessageBox( NULL, str, "Backup",
        MB_YESNO|MB_ICONQUESTION ) == IDYES )
    {
        return 1;
    }
    return 0;
}
extern short _callback MessageCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback StatusCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback ErrorCallBack( char * str )
{

```

```

    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
typedef void (CALLBACK *DBTOOLSPROC)( void * );
typedef short (CALLBACK *DBTOOLSFUNC)( void * );

// Main entry point into the program.
int main( int argc, char * argv[] )
{
    a_dbtools_info  dbt_info;
    a_backup_db     backup_info;
    char            dir_name[ _MAX_PATH + 1];
    char            connect[ 256 ];
    HINSTANCE       hinst;
    DBTOOLSFUNC     dbbackup;
    DBTOOLSFUNC     dbtoolsinit;
    DBTOOLSPROC     dbtoolsfini;
    short           ret_code;

    // Always initialize to 0 so new versions
    // of the structure will be compatible.
    memset( &dbt_info, 0, sizeof( a_dbtools_info ) );
    dbt_info.errorrtn = (MSG_CALLBACK)MessageCallBack;;

    memset( &backup_info, 0, sizeof( a_backup_db ) );
    backup_info.version = DB_TOOLS_VERSION_NUMBER;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.confirmrtn = (MSG_CALLBACK)ConfirmCallBack;
    backup_info.errorrtn = (MSG_CALLBACK)ErrorCallBack;
    backup_info.msgrtn = (MSG_CALLBACK)MessageCallBack;
    backup_info.statusrtn = (MSG_CALLBACK)StatusCallBack;
    if( argc > 1 )
    {
        strncpy( dir_name, argv[1], _MAX_PATH );
    }
    else
    {
        // DBTools does not expect (or like) a trailing slash
        strcpy( dir_name, "c:\\temp" );
    }
    backup_info.output_dir = dir_name;
    if( argc > 2 )
    {
        strncpy( connect, argv[2], 255 );
    }
    else
    {
        strcpy( connect, "DSN=Sybase IQ Demo" );
    }
    backup_info.connectparms = connect;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;

```

Database Tools Interface (DBTools)

```
backup_info.backup_database = 1;
backup_info.backup_logfile = 1;
backup_info.rename_log = 0;
backup_info.truncate_log = 0;
hinst = LoadLibrary( "dbtool16.dll" );
if( hinst == NULL )
{
    // Failed
    return EXIT_FAIL;
}
dbbackup = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
    "_DBBackup@4" );
dbtoolsinit = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
    "_DBToolsInit@4" );
dbtoolsfini = (DBTOOLSPROC) GetProcAddress( (HMODULE)hinst,
    "_DBToolsFini@4" );
ret_code = (*dbtoolsinit)( &dbt_info );
if( ret_code != EXIT_OKAY ) {
    return ret_code;
}
ret_code = (*dbbackup)( &backup_info );
(*dbtoolsfini)( &dbt_info );
FreeLibrary( hinst );
return ret_code;
}
```

Software Component Exit Codes

All database tools library entry points use the following exit codes. The SAP Sybase IQ utilities also use these exit codes.

Code	Status	Explanation
0	EXIT_OKAY	Success
1	EXIT_FAIL	General failure
2	EXIT_BAD_DATA	Invalid file format
3	EXIT_FILE_ERROR	File not found, unable to open
4	EXIT_OUT_OF_MEMORY	Out of memory
5	EXIT_BREAK	Terminated by the user
6	EXIT_COMMUNICATIONS_FAIL	Failed communications
7	EXIT_MISSING_DATABASE	Missing a required database name

Code	Status	Explanation
8	EXIT_PROTOCOL_MISMATCH	Client/server protocol mismatch
9	EXIT_UNABLE_TO_CONNECT	Unable to connect to the database server
10	EXIT_ENGINE_NOT_RUNNING	Database server not running
11	EXIT_SERVER_NOT_FOUND	Database server not found
12	EXIT_BAD_ENCRYPT_KEY	Missing or bad encryption key
13	EXIT_DB_VER_NEWER	Server must be upgraded to run database
14	EXIT_FILE_INVALID_DB	File is not a database
15	EXIT_LOG_FILE_ERROR	Log file was missing or other error
16	EXIT_FILE_IN_USE	File in use
17	EXIT_FATAL_ERROR	Fatal error occurred
18	EXIT_MISSING_LICENSE_FILE	Missing server license file
19	EXIT_BACKGROUND_SYNC_ABORTED	Background synchronization aborted to allow higher priority operations proceed
20	EXIT_FILE_ACCESS_DENIED	Database cannot be started because access is denied
255	EXIT_USAGE	Invalid parameters on the command line
21	EXIT_SERVER_NAME_IN_USE	Another server with the same name is currently running.

These exit codes are defined in the %IQDIR16%\sdk\include\sqldef.h file.

Database Tools C API Reference

The header files are `dbtools.h` and `dbrmt.h`.

DBBackup(const a_backup_db *) method

Backs up a database.

Syntax

```
_crtn short _entry DBBackup(const a_backup_db * pdb)
```

Parameters

- **pdb** – Pointer to a properly initialized a_backup_db structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the dbbackup utility.

The DBBackup function manages all client-side database backup tasks. For a description of these tasks, see Backup utility (dbbackup).

To perform a server-side backup, use the BACKUP DATABASE statement.

DBChangeLogName(const a_change_log *) method

Changes the name of the transaction log file.

Syntax

```
_crtn short _entry DBChangeLogName(const a_change_log * pcl)
```

Parameters

- **pcl** – Pointer to a properly initialized a_change_log structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the dblog utility.

The -t option of the Transaction Log utility (dblog) changes the name of the transaction log. DBChangeLogName provides a programmatic interface to this function.

DBCCreate(a_create_db *) method

Creates a database.

Syntax

```
_crtn short _entry DBCreate(a_create_db * pcdb)
```

Parameters

- **pcdb** – Pointer to a properly initialized a_create_db structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the dbinit utility.

DBCreatedVersion(a_db_version_info *) method

Determines the version of SQL Anywhere that was used to create a database file, without attempting to start the database.

Syntax

```
_crtn short _entry DBCreatedVersion(a_db_version_info * pdvi)
```

Parameters

- **pdvi** – Pointer to a properly initialized a_db_version_info structure.

Returns

Return code, as listed in Software component exit codes.

Usage

Currently, this function only differentiates between databases built with version 9 or earlier and those built with version 10 or later.

Version information is not set if a failing code is returned.

DBErase(const an_erase_db *) method

Erases a database file and/or transaction log file.

Syntax

```
_crtn short _entry DBErase(const an_erase_db * pedb)
```

Parameters

- **pedb** – Pointer to a properly initialized `an_erase_db` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the `dberase` utility.

DBInfo(a_db_info *) method

Returns information about a database file.

Syntax

```
_crtn short _entry DBInfo(a_db_info * pdbi)
```

Parameters

- **pdbi** – Pointer to a properly initialized `a_db_info` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the `dbinfo` utility.

DBInfoDump(a_db_info *) method

Returns information about a database file.

Syntax

```
_crtn short _entry DBInfoDump( a_db_info * pdbi)
```

Parameters

- **pdbi** – Pointer to a properly initialized `a_db_info` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the `dbinfo` utility when the `-u` option is used.

DBInfoFree(a_db_info *) method

Frees resources after the DBInfoDump function is called.

Syntax

```
_crtm short _entry DBInfoFree(a_db_info * pdbi)
```

Parameters

- **pdbi** – Pointer to a properly initialized a_db_info structure.

Returns

Return code, as listed in Software component exit codes.

DBLicense(const a_dblic_info *) method

Modifies or reports the licensing information of the database server.

Syntax

```
_crtm short _entry DBLicense(const a_dblic_info * pdi)
```

Parameters

- **pdi** – Pointer to a properly initialized a_dblic_info structure.

Returns

Return code, as listed in Software component exit codes.

DBLogFileInfo(const a_log_file_info *) method

Returns the log file and mirror log file paths of a non-running database file.

Syntax

```
_crtm short _entry DBLogFileInfo(const a_log_file_info * plfi)
```

Parameters

- **plfi** – Pointer to a properly initialized a_log_file_info structure.

Returns

Return code, as listed in Software component exit codes.

Usage

Note that this function will only work for databases that have been created with SQL Anywhere 10.0.0 and up.

DBRemoteSQL(a_remote_sql *) method

Accesses the SQL Remote Message Agent.

Syntax

```
_crtn short _entry DBRemoteSQL( a_remote_sql * prs)
```

Parameters

- **prs** – Pointer to a properly initialized a_remote_sql structure.

Returns

Return code, as listed in Software component exit codes.

Usage

For information about the features you can access, see SQL Remote Message Agent utility (dbremote).

DBSynchronizeLog(const a_sync_db *) method

Synchronize a database with a MobiLink server.

Syntax

```
_crtn short _entry DBSynchronizeLog(const a_sync_db * psdb)
```

Parameters

- **psdb** – Pointer to a properly initialized a_sync_db structure.

Returns

Return code, as listed in Software component exit codes.

DBToolsFini(const a_dbtools_info *) method

Decrements a reference counter and frees resources when an application is finished with the DBTools library.

Syntax

```
_crtn short _entry DBToolsFini(const a_dbtools_info * pdi)
```

Parameters

- **pdi** – Pointer to a properly initialized `a_dbtools_info` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

The `DBToolsFini` function must be called at the end of any application that uses the DBTools interface. Failure to do so can lead to lost memory resources.

DBToolsInit(const a_dbtools_info *) method

Prepares the DBTools library for use.

Syntax

```
_crtn short _entry DBToolsInit(const a_dbtools_info * pdi)
```

Parameters

- **pdi** – Pointer to a properly initialized `a_dbtools_info` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

The primary purpose of the `DBToolsInit` function is to load the SQL Anywhere messages library. The messages library contains localized versions of error messages and prompts used by the functions in the DBTools library.

The `DBToolsInit` function must be called at the start of any application that uses the DBTools interface, before any other DBTools functions.

DBToolsVersion(void) method

Returns the version number of the DBTools library.

Syntax

```
_crtn short _entry DBToolsVersion(void)
```

Usage

Use the `DBToolsVersion` function to check that the DBTools library is not older than one against which your application is developed. While applications can run against newer versions of DBTools, they cannot run against older versions.

DBTranslateLog(const a_translate_log *) method

Translates a transaction log file to SQL.

Syntax

```
_crtn short _entry DBTranslateLog(const a_translate_log * ptl)
```

Parameters

- **ptl** – Pointer to a properly initialized a_translate_log structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the dbtran utility.

DBTruncateLog(const a_truncate_log *) method

Truncates a transaction log file.

Syntax

```
_crtn short _entry DBTruncateLog(const a_truncate_log * ptl)
```

Parameters

- **ptl** – Pointer to a properly initialized a_truncate_log structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the dbbackup utility.

DBUnload(an_unload_db *) method

Unloads a database.

Syntax

```
_crtn short _entry DBUnload( an_unload_db * pudb)
```

Parameters

- **puodb** – Pointer to a properly initialized `an_unload_db` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the `dbunload` and `dbextract` utilities.

DBUpgrade(const an_upgrade_db *) method

Upgrades a database file.

Syntax

```
_crtn short _entry DBUpgrade(const an_upgrade_db * puodb)
```

Parameters

- **puodb** – Pointer to a properly initialized `an_upgrade_db` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the `dbupgrad` utility.

DBValidate(const a_validate_db *) method

Validates all or part of a database.

Syntax

```
_crtn short _entry DBValidate(const a_validate_db * pvdb)
```

Parameters

- **pvdb** – Pointer to a properly initialized `a_validate_db` structure.

Returns

Return code, as listed in Software component exit codes.

Usage

This function is used by the `dbvalid` utility.

Caution: Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, spurious errors may be reported indicating some form of database corruption even though no corruption actually exists.

Autotune() enumeration

Used in the `a_backup_db` structure to control auto tuning of writers.

Enum Constant Summary

- **BACKUP_AUTO_TUNE_UNSPECIFIED** – Use to leave AUTO TUNE WRITERS clause unspecified.
- **BACKUP_AUTO_TUNE_ON** – Use to generate AUTO TUNE WRITERS ON clause.
- **BACKUP_AUTO_TUNE_OFF** – Use to generate AUTO TUNE WRITERS OFF clause.

Checkpoint() enumeration

Used in the `a_backup_db` structure to control copying of the checkpoint log.

Enum Constant Summary

- **BACKUP_CHKPT_LOG_COPY** – Use to generate WITH CHECKPOINT LOG COPY clause.
- **BACKUP_CHKPT_LOG_NOCOPY** – Use to generate WITH CHECKPOINT LOG NOCOPY clause.
- **BACKUP_CHKPT_LOG_RECOVER** – Use to generate WITH CHECKPOINT LOG RECOVER clause.
- **BACKUP_CHKPT_LOG_AUTO** – Use to generate WITH CHECKPOINT LOG AUTO clause.
- **BACKUP_CHKPT_LOG_DEFAULT** – Use to omit WITH CHECKPOINT clause.

History() enumeration

Used in the `a_backup_db` structure to control enabling of backup history.

Enum Constant Summary

- **BACKUP_HISTORY_UNSPECIFIED** – Use to leave HISTORY clause unspecified.
- **BACKUP_HISTORY_ON** – Use to generate HISTORY ON clause.
- **BACKUP_HISTORY_OFF** – Use to generate HISTORY OFF clause.

Padding() enumeration

Blank padding enumeration specifies the blank_pad setting in a_create_db.

Enum Constant Summary

- **NO_BLANK_PADDING** – Does not use blank padding.
- **BLANK_PADDING** – Uses blank padding.

Unit() enumeration

Used in the a_create_db structure, to specify the value of db_size_unit.

Enum Constant Summary

- **DBSP_UNIT_NONE** – Units not specified.
- **DBSP_UNIT_PAGES** – Size is specified in pages.
- **DBSP_UNIT_BYTES** – Size is specified in bytes.
- **DBSP_UNIT_KILOBYTES** – Size is specified in kilobytes.
- **DBSP_UNIT_MEGABYTES** – Size is specified in megabytes.
- **DBSP_UNIT_GIGABYTES** – Size is specified in gigabytes.
- **DBSP_UNIT_TERABYTES** – Size is specified in terabytes.

Unload() enumeration

The type of unload being performed, as used by the an_unload_db structure.

Enum Constant Summary

- **UNLOAD_ALL** – Unload both data and schema.
- **UNLOAD_DATA_ONLY** – Unload data. Do not unload schema. Equivalent to dbunload -d option.
- **UNLOAD_NO_DATA** – No data. Unload schema only. Equivalent to dbunload -n option.
- **UNLOAD_NO_DATA_FULL_SCRIPT** – No data. Include LOAD/INPUT statements in reload script. Equivalent to dbunload -nl option.
- **UNLOAD_NO_DATA_NAME_ORDER** – No data. Objects will be output ordered by name.

UserList() enumeration

The type of a user list, as used by an_a_translate_log structure.

Enum Constant Summary

- **DBTRAN_INCLUDE_ALL** – Include operations from all users.

- **DBTRAN_INCLUDE_SOME** – Include operations only from the users listed in the supplied user list.
- **DBTRAN_EXCLUDE_SOME** – Exclude operations from the users listed in the supplied user list.

Validation() enumeration

The type of validation being performed, as used by the `a_validate_db` structure.

Enum Constant Summary

- **VALIDATE_NORMAL** – Validate with the default check only.
- **VALIDATE_DATA** – (obsolete)
- **VALIDATE_INDEX** – (obsolete)
- **VALIDATE_EXPRESS** – Validate with express check. Equivalent to `dbvalid -fx` option.
- **VALIDATE_FULL** – (obsolete)
- **VALIDATE_CHECKSUM** – Validate database checksums. Equivalent to `dbvalid -s` option.
- **VALIDATE_DATABASE** – Validate database. Equivalent to `dbvalid -d` option.
- **VALIDATE_COMPLETE** – Perform all possible validation activities.

Verbosity() enumeration

Verbosity enumeration specifies the volume of output.

Enum Constant Summary

- **VB_QUIET** – No output.
- **VB_NORMAL** – Normal amount of output.
- **VB_VERBOSE** – Verbose output, useful for debugging.

Version() enumeration

Used in the `a_db_version_info` structure, to indicate the version of SQL Anywhere that initially created the database.

Enum Constant Summary

- **VERSION_UNKNOWN** – Unable to determine the version of SQL Anywhere that created the database.
- **VERSION_PRE_10** – Database was created using SQL Anywhere version 9 or earlier.
- **VERSION_10** – Database was created using SQL Anywhere version 10.
- **VERSION_11** – Database was created using SQL Anywhere version 11.
- **VERSION_12** – Database was created using SQL Anywhere version 12.
- **VERSION_16** – Database was created using SQL Anywhere version 16.

a_backup_db structure

Holds the information needed to perform backup tasks using the DBTools library.

Syntax

```
typedef struct a_backup_db
```

auto_tune_writers char

Enable/disable auto tune writers.

Syntax

```
public char auto_tune_writers;
```

Remarks

Must be one of BACKUP_AUTO_TUNE_UNSPECIFIED, BACKUP_AUTO_TUNE_ON, or BACKUP_AUTO_TUNE_OFF. Use to generate AUTO TUNE WRITERS OFF clause. Set by dbbackup -aw[-] option

backup_comment const char *

Comment used for the WITH COMMENT clause.

Syntax

```
public const char * backup_comment;
```

backup_database a_bit_field

Back up the database file.

Syntax

```
public a_bit_field backup_database;
```

Remarks

Set TRUE by dbbackup -d option.

backup_history char

Backup history.

Syntax

```
public char backup_history;
```

Remarks

Must be one of BACKUP_HISTORY_UNSPECIFIED, BACKUP_HISTORY_ON, or BACKUP_HISTORY_OFF. Set by dbbackup -h[-] option

backup_interrupted char

Indicates that the operation was interrupted when non-zero.

Syntax

```
public char backup_interrupted;
```

backup_logfile a_bit_field

Back up the transaction log file.

Syntax

```
public a_bit_field backup_logfile;
```

Remarks

Set TRUE by dbbackup -t option.

chkpt_log_type char

Control copying of checkpoint log.

Syntax

```
public char chkpt_log_type;
```

Remarks

Must be one of BACKUP_CHKPT_LOG_COPY, BACKUP_CHKPT_LOG_NOCOPY, BACKUP_CHKPT_LOG_RECOVER, BACKUP_CHKPT_LOG_AUTO, or BACKUP_CHKPT_LOG_DEFAULT. Set by dbbackup -k option.

confirmrtn MSG_CALLBACK

Address of a confirmation request callback routine or NULL.

Syntax

```
public MSG_CALLBACK confirmrtn;
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:
"UID=DBA;PWD=sql;DBF=demo.db".

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

hotlog_filename const char *

File name for the live backup file.

Syntax

```
public const char * hotlog_filename;
```

Remarks

Set by dbbackup -l option.

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

no_confirm a bit field

Operate without confirmation.

Syntax

```
public a_bit_field no_confirm;
```

Remarks

Set TRUE by dbbackup -y option.

output_dir const char *

Path to the output directory for backups, for example: "c:\backup".

Syntax

```
public const char * output_dir;
```

page_blocksize a_sql_uint32

Number of pages in data blocks.

Syntax

```
public a_sql_uint32 page_blocksize;
```

Remarks

If set to 0, then the default is 128. Set by dbbackup -b option.

progress_messages a_bit_field

Display progress messages.

Syntax

```
public a_bit_field progress_messages;
```

Remarks

Set TRUE by dbbackup -p option.

quiet a_bit_field

Operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dbbackup -q option.

rename_local_log a_bit_field

Rename the local backup of the transaction log.

Syntax

```
public a_bit_field rename_local_log;
```

Remarks

Set TRUE by dbbackup -n option.

rename_log a_bit_field

Rename the transaction log.

Syntax

```
public a_bit_field rename_log;
```

Remarks

Set TRUE by dbbackup -r option.

server_backup a_bit_field

Perform backup on server using BACKUP DATABASE.

Syntax

```
public a_bit_field server_backup;
```

Remarks

Set TRUE by dbbackup -s option.

statusrtn MSG_CALLBACK

Address of a status message callback routine or NULL.

Syntax

```
public MSG_CALLBACK statusrtn;
```

truncate_log a_bit_field

Delete the transaction log.

Syntax

```
public a_bit_field truncate_log;
```

Remarks

Set TRUE by dbbackup -x option.

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

wait_after_end a_bit_field

Wait after end.

Syntax

```
public a_bit_field wait_after_end;
```

Remarks

Set TRUE by dbbackup -wa option.

wait_before_start a_bit_field

Wait before start.

Syntax

```
public a_bit_field wait_before_start;
```

Remarks

Set TRUE by dbbackup -wb option.

a_change_log structure

Holds the information needed to perform dblog tasks using the DBTools library.

Syntax

```
typedef struct a_change_log
```

change_logname a_bit_field

Set TRUE to permit changing of the transaction log name.

Syntax

```
public a_bit_field change_logname;
```

Remarks

Set TRUE by dblog -n or -t option.

change_mirrorname a_bit_field

Set TRUE to permit changing of the mirror log name.

Syntax

```
public a_bit_field change_mirrorname;
```

Remarks

Set TRUE by dblog -m, -n, or -r option.

dbname const char *

Database file name.

Syntax

```
public const char * dbname;
```

encryption_key char *

The encryption key for the database file. Equivalent to `dblog -ek` or `-ep` option.

Syntax

```
public char * encryption_key;
```

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

generation_number unsigned short

The new generation number. Reserved, use zero.

Syntax

```
public unsigned short generation_number;
```

ignore_dbsync_trunc a bit field

When using `dbmsync`, resets the offset kept for the `delete_old_logs` option, allowing transaction logs to be deleted when they are no longer needed.

Syntax

```
public a_bit_field ignore_dbsync_trunc;
```

Remarks

Set TRUE by `dblog -is` option.

ignore_ltm_trunc a bit field

Reserved, use FALSE.

Syntax

```
public a_bit_field ignore_ltm_trunc;
```

ignore_remote_trunc a bit field

For SQL Remote.

Syntax

```
public a_bit_field ignore_remote_trunc;
```

Remarks

Resets the offset kept for the `delete_old_logs` option, allowing transaction logs to be deleted when they are no longer needed. Set TRUE by `dblog -ir` option.

logname const char *

Transaction log file name, or NULL if there is no log.

Syntax

```
public const char * logname;
```

mirrorname const char *

The new name of the transaction log mirror file. Equivalent to dblog -m option.

Syntax

```
public const char * mirrorname;
```

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

query_only a bit field

If 1, just display the name of the transaction log. If 0, permit changing of the log name.

Syntax

```
public a_bit_field query_only;
```

quiet a bit field

Operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dblog -q option.

set_generation_number a bit field

Reserved. Use FALSE.

Syntax

```
public a_bit_field set_generation_number;
```


version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

zap_current_offset char *

Change the current offset to the specified value.

Syntax

```
public char * zap_current_offset;
```

Remarks

This is for use only in resetting a transaction log after an unload and reload to match dbremote or dbmsync settings. Equivalent to dblog -x option.

zap_starting_offset char *

Change the starting offset to the specified value.

Syntax

```
public char * zap_starting_offset;
```

Remarks

This is for use only in resetting a transaction log after an unload and reload to match dbremote or dbmsync settings. Equivalent to dblog -z option.

a_create_db structure

Holds the information needed to create a database using the DBTools library.

Syntax

```
typedef struct a_create_db
```

accent_sensitivity char

One of 'y', 'n', or 'f' (yes, no, French).

Syntax

```
public char accent_sensitivity;
```

Remarks

Generates one of the ACCENT RESPECT, ACCENT IGNORE or ACCENT FRENCH clauses.

avoid_view_collisions a_bit_field

Set TRUE to omit the generation of Watcom SQL compatibility views SYS.SYSCOLUMNS and SYS.SYSINDEXES.

Syntax

```
public a_bit_field avoid_view_collisions;
```

Remarks

Set TRUE by dbinit -k option.

blank_pad a_bit_field

Must be one of NO_BLANK_PADDING or BLANK_PADDING.

Syntax

```
public a_bit_field blank_pad;
```

Remarks

Treat blanks as significant in string comparisons and hold index information to reflect this. See Blank padding enumeration. Equivalent to dbinit -b option.

case_sensitivity_use_default a_bit_field

Set TRUE to use the default case sensitivity for the locale.

Syntax

```
public a_bit_field case_sensitivity_use_default;
```

Remarks

This only affects UCA. If set TRUE then we do not add the CASE RESPECT clause to the CREATE DATABASE statement.

checksum a_bit_field

Set to TRUE for ON or FALSE for OFF.

Syntax

```
public a_bit_field checksum;
```

Remarks

Generates one of CHECKSUM ON or CHECKSUM OFF clauses. Set TRUE by dbinit -s option.

data_store_type const char *

Reserved. Use NULL.

Syntax

```
public const char * data_store_type;
```

db_size unsigned int

When not 0, generates the DATABASE SIZE clause. Equivalent to dbinit -dbs option.

Syntax

```
public unsigned int db_size;
```

db_size_unit int

Used with db_size, must be one of DBSP_UNIT_NONE, DBSP_UNIT_PAGES, DBSP_UNIT_BYTES, DBSP_UNIT_KILOBYTES, DBSP_UNIT_MEGABYTES, DBSP_UNIT_GIGABYTES, or DBSP_UNIT_TERABYTES.

Syntax

```
public int db_size_unit;
```

Remarks

When not DBSP_UNIT_NONE, it generates the corresponding keyword (for example, DATABASE SIZE 10 MB is generated when db_size is 10 and db_size_unit is DBSP_UNIT_MEGABYTES). See Database size unit enumeration.

dba_pwd char *

When not NULL, generates the DBA PASSWORD xxx clause. Equivalent to dbinit -dba option.

Syntax

```
public char * dba_pwd;
```

dba_uid char *

When not NULL, generates the DBA USER xxx clause. Equivalent to dbinit -dba option.

Syntax

```
public char * dba_uid;
```

dbname const char *

Database file name.

Syntax

```
public const char * dbname;
```

default_collation const char *

The collation for the database. Equivalent to dbinit -z option.

Syntax

```
public const char * default_collation;
```

encoding const char *

The character set encoding. Equivalent to dbinit -ze option.

Syntax

```
public const char * encoding;
```

encrypt a bit field

Set TRUE to generate the ENCRYPTED ON clause or, when encrypted_tables is also set, the ENCRYPTED TABLES ON clause.

Syntax

```
public a_bit_field encrypt;
```

Remarks

Set TRUE by dbinit -e? options.

encrypted_tables a bit field

Set TRUE to encrypt tables.

Syntax

```
public a_bit_field encrypted_tables;
```

Remarks

Used with encrypt, it generates the ENCRYPTED TABLE ON clause instead of the ENCRYPTED ON clause. Set TRUE by dbinit -et option.

encryption_algorithm const char *

The encryption algorithm (AES, AES256, AES_FIPS, or AES256_FIPS).

Syntax

```
public const char * encryption_algorithm;
```

Remarks

Used with encrypt and encryption_key, it generates the ALGORITHM clause. Equivalent to dbinit -ea option.

encryption_key const char *

The encryption key for the database file.

Syntax

```
public const char * encryption_key;
```

Remarks

Used with encrypt, it generates the KEY clause. Equivalent to dbinit -ek option.

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

iq_params void *

Reserved. Use NULL.

Syntax

```
public void * iq_params;
```

jconnect a_bit_field

Set TRUE to include system procedures needed for jConnect.

Syntax

```
public a_bit_field jconnect;
```

Remarks

Set FALSE by dbinit -i option.

logname const char *

New transaction log name. Equivalent to dbinit -t option.

Syntax

```
public const char * logname;
```

mirrorname const char *

Transaction log mirror name. Equivalent to dbinit -m option.

Syntax

```
public const char * mirrorname;
```

msgtrn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msgtrn;
```

nchar_collation const char *

The NCHAR COLLATION for the database when not NULL. Equivalent to dbinit -zn option.

Syntax

```
public const char * nchar_collation;
```

page_size unsigned short

The page size of the database. Equivalent to dbinit -p option.

Syntax

```
public unsigned short page_size;
```

respect_case a_bit_field

Make string comparisons case sensitive and hold index information to reflect this.

Syntax

```
public a_bit_field respect_case;
```

Remarks

Set TRUE by dbinit -c option.

startline const char *

The command line used to start the database server.

Syntax

```
public const char * startline;
```

Remarks

For example: "c:\SQLAny16\bin32\dbsrv16.exe". If NULL, the default START parameter is "dbeng16 -gp <page_size> -c 10M" for SQL Anywhere where page_size is specified below. Note that "-c 10M" is appended if page_size >= 2048.

sys_proc_definer a_bit_field

Set TRUE to retain the SQL SECURITY Model for version 12.0.1 or earlier system stored procedures.

Syntax

```
public a_bit_field sys_proc_definer;
```

Remarks

Set TRUE by dbinit -pd option.

verbose char

See Verbosity enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).

Syntax

```
public char verbose;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_db_info structure

Holds the information needed to return DBInfo information using the DBTools library.

Syntax

```
typedef struct a_db_info
```

bit_map_pages a_sql_uint32

Number of bitmap pages in the database.

Syntax

```
public a_sql_uint32 bit_map_pages;
```

charcollationspecbuffer char *

Pointer to the char collation string buffer.

Syntax

```
public char * charcollationspecbuffer;
```

charcollationspecbufsize unsigned short

Size of charcollationspecbuffer (at least 256+1).

Syntax

```
public unsigned short charcollationspecbufsize;
```

charencodingbuffer char *

Pointer to the char encoding string buffer.

Syntax

```
public char * charencodingbuffer;
```

charencodingbufsize unsigned short

Size of charencodingbuffer (at least 50+1).

Syntax

```
public unsigned short charencodingbufsize;
```

checksum a bit field

If set TRUE, global checksums are enabled (a checksum on every database page).

Syntax

```
public a_bit_field checksum;
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:

```
"UID=DBA;PWD=sql;DBF=demo.db".
```

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```


dbbufsize unsigned short

Size of dbnamebuffer (for example, _MAX_PATH).

Syntax

```
public unsigned short dbbufsize;
```

dbnamebuffer char *

Pointer to the database file name buffer.

Syntax

```
public char * dbnamebuffer;
```

encrypted_tables a_bit_field

If set TRUE, encrypted tables are supported.

Syntax

```
public a_bit_field encrypted_tables;
```

errortn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errortn;
```

file_size a_sql_uint32

Size of database file (in pages).

Syntax

```
public a_sql_uint32 file_size;
```

free_pages a_sql_uint32

Number of free pages.

Syntax

```
public a_sql_uint32 free_pages;
```

logbufsize unsigned short

Size of lognamebuffer (for example, _MAX_PATH).

Syntax

```
public unsigned short logbufsize;
```

lognamebuffer char *

Pointer to the transaction log file name buffer.

Syntax

```
public char * lognamebuffer;
```

mirrorbufsize unsigned short

Size of mirrornamebuffer (for example, _MAX_PATH).

Syntax

```
public unsigned short mirrorbufsize;
```

mirrornamebuffer char *

Pointer to the mirror file name buffer.

Syntax

```
public char * mirrornamebuffer;
```

msgtrn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msgtrn;
```

ncharcollationspecbuffer char *

Pointer to the nchar collation string buffer.

Syntax

```
public char * ncharcollationspecbuffer;
```

ncharcollationspecbufsize unsigned short

Size of ncharcollationspecbuffer (at least 256+1).

Syntax

```
public unsigned short ncharcollationspecbufsize;
```

ncharencodingbuffer char *

Pointer to the nchar encoding string buffer.

Syntax

```
public char * ncharencodingbuffer;
```

ncharencodingbufsize unsigned short

Size of ncharencodingbuffer (at least 50+1).

Syntax

```
public unsigned short ncharencodingbufsize;
```

other_pages a_sql_uint32

Number of pages that are not table pages, index pages, free pages, or bitmap pages.

Syntax

```
public a_sql_uint32 other_pages;
```

page_usage a_bit_field

Set TRUE to report page usage statistics, otherwise FALSE.

Syntax

```
public a_bit_field page_usage;
```

Remarks

Set TRUE by dbinfo -u option.

quiet a_bit_field

Set TRUE to operate without confirming messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dbinfo -q option.

statusrtn MSG_CALLBACK

Address of a status message callback routine or NULL.

Syntax

```
public MSG_CALLBACK statusrtn;
```

sysinfo a_sysinfo

Inline a_sysinfo structure.

Syntax

```
public a_sysinfo sysinfo;
```

totals a_table_info *

Pointer to a_table_info structure.

Syntax

```
public a_table_info * totals;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_db_version_info structure

Holds information regarding which version of SQL Anywhere was used to create the database.

Syntax

```
typedef struct a_db_version_info
```

created_version char

Set to one of VERSION_UNKNOWN, VERSION_PRE_10, etc.

Syntax

```
public char created_version;
```

Remarks

indicating the server version that created the database file.

errortn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errortn;
```

filename const char *

Name of the database file to check.

Syntax

```
public const char * filename;
```

msg rtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msg rtn;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_dblic_info structure

Holds information containing licensing information.

Syntax

```
typedef struct a_dblic_info
```

Remarks

You must use this information only in a manner consistent with your license agreement.

compname char *

Company name for licensing.

Syntax

```
public char * compname;
```

conncount a_sql_int32

Maximum number of connections licensed.

Syntax

```
public a_sql_int32 conncount;
```

Remarks

To set, use 1000000L for default.

error rtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK error rtn;
```

exename char *

Name of the server executable or license file.

Syntax

```
public char * exename;
```

installkey char *

Reserved; set NULL.

Syntax

```
public char * installkey;
```

Remarks

Set by dblic -k option.

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

nodecount a_sql_int32

Number of nodes licensed.

Syntax

```
public a_sql_int32 nodecount;
```

query_only a_bit_field

Set TRUE to just display the license information.

Syntax

```
public a_bit_field query_only;
```

Remarks

Set FALSE to permit changing the information.

quiet a_bit_field

Set TRUE to operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dblic -q option.

type a_license_type

See lictype.h for values.

Syntax

```
public a_license_type type;
```

Remarks

One of LICENSE_TYPE_PERSEAT, LICENSE_TYPE_CONCURRENT, or LICENSE_TYPE_PERCPU.

username char *

User name for licensing.

Syntax

```
public char * username;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_dbtools_info structure

DBTools information callback used to initialize and finalize the DBTools library calls.

Syntax

```
typedef struct a_dbtools_info
```

errortn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errortn;
```

a_log_file_info structure

Used to obtain the log file and mirror log file information of a non-running database.

Syntax

```
typedef struct a_log_file_info
```

dbname const char *

Database file name.

Syntax

```
public const char * dbname;
```

encryption_key const char *

The encryption key for the database file.

Syntax

```
public const char * encryption_key;
```

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

logname char *

Buffer for transaction log file name, or NULL.

Syntax

```
public char * logname;
```

logname_size size_t

Size of buffer for transaction log file name, or zero.

Syntax

```
public size_t logname_size;
```

mirrorname char *

Buffer for mirror log file name, or NULL.

Syntax

```
public char * mirrorname;
```

mirrorname_size size_t

Size of buffer for mirror log file name, or zero.

Syntax

```
public size_t mirrorname_size;
```


reserved void *

Reserved for internal use and must set to NULL.

Syntax

```
public void * reserved;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_name structure

Specifies a variable list of names.

Syntax

```
typedef struct a_name
```

name char

One or more bytes comprising the name.

Syntax

```
public char name;
```

next struct a_name *

Pointer to the next name in the list or NULL.

Syntax

```
public struct a_name * next;
```

a_remote_sql structure

Holds information needed for the dbremote utility using the DBTools library.

Syntax

```
typedef struct a_remote_sql
```

Remarks

The dbremote utility sets the following defaults before processing any command-line options:

- version = DB_TOOLS_VERSION_NUMBER
- argv = (argument vector passed to application)
- deleted = TRUE

Database Tools Interface (DBTools)

- `apply = TRUE`
- `more = TRUE`
- `link_debug = FALSE`
- `max_length = 50000`
- `memory = 2 * 1024 * 1024`
- `frequency = 1`
- `threads = 0`
- `receive_delay = 60`
- `send_delay = 0`
- `log_size = 0`
- `patience_retry = 1`
- `resend_urgency = 0`
- `log_file_name = (set from command line)`
- `truncate_remote_output_file = FALSE`
- `remote_output_file_name = NULL`
- `no_user_interaction = TRUE` (if user interface is not available)
- `errorrtn = (address of an appropriate routine)`
- `msggrtn = (address of an appropriate routine)`
- `confirmrtn = (address of an appropriate routine)`
- `msgqueuertn = (address of an appropriate routine)`
- `logrtn = (address of an appropriate routine)`
- `warningrtn = (address of an appropriate routine)`
- `set_window_title_rtn = (address of an appropriate routine)`
- `progress_msg_rtn = (address of an appropriate routine)`
- `progress_index_rtn = (address of an appropriate routine)`

apply a_bit_field

Normally set TRUE.

Syntax

```
public a_bit_field apply;
```

Remarks

When not set, messages are scanned but not applied. Corresponds to `dbremote -a` option.

argv char **

Pointer to a parsed command line (a vector of pointers to strings).

Syntax

```
public char ** argv;
```

Remarks

If not NULL, then DBRemoteSQL will call a message routine to display each command line argument except those prefixed with -c, -cq, or -ek.

batch a_bit_field

When set TRUE, force exit after applying message and scanning log (this is the same as at least one user having 'always' send time).

Syntax

```
public a_bit_field batch;
```

Remarks

When cleared, allow run mode to be determined by remote users send times.

confirmrtn MSG_CALLBACK

Address of a confirmation request callback routine or NULL.

Syntax

```
public MSG_CALLBACK confirmrtn;
```

connectparms char *

Parameters needed to connect to the database.

Syntax

```
public char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:
"UID=DBA;PWD=sql;DBF=demo.db".

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbeng16.exe".

A full example connection string including the START parameter:
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbeng16.exe".

debug a_bit_field

When set TRUE, debug output is included.

Syntax

```
public a_bit_field debug;
```

debug_dump_size a_sql_uint32

Reserved for internal use and must set to 0.

Syntax

```
public a_sql_uint32 debug_dump_size;
```

debug_page_offsets a_bit_field

Reserved for internal use and must set to FALSE.

Syntax

```
public a_bit_field debug_page_offsets;
```

default_window_title char *

A pointer to the default window title string.

Syntax

```
public char * default_window_title;
```

deleted a_bit_field

Normally set TRUE.

Syntax

```
public a_bit_field deleted;
```

Remarks

When not set, messages are not deleted after they are applied. Corresponds to dbremote -p option.

encryption_key char *

Pointer to an encryption key. Corresponds to the dbremote -ek option.

Syntax

```
public char * encryption_key;
```

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

frequency a_sql_uint32

Reserved for internal use and must set to 0.

Syntax

```
public a_sql_uint32 frequency;
```

full_q_scan a_bit_field

Reserved for internal use and must set to FALSE.

Syntax

```
public a_bit_field full_q_scan;
```

include_scan_range char *

Reserved for internal use and must set to NULL.

Syntax

```
public char * include_scan_range;
```

latest_backup a_bit_field

When set TRUE, only logs that are backed up are processed.

Syntax

```
public a_bit_field latest_backup;
```

Remarks

Don't send operations from a live log. Corresponds to the dbremote -u option.

link_debug a_bit_field

When set TRUE, debugging will be turned on for links.

Syntax

```
public a_bit_field link_debug;
```

locale char *

Reserved for internal use and must set to NULL.

Syntax

```
public char * locale;
```

log_file_name const char *

Pointer to the name of the DBRemoteSQL output log to which the message callbacks print their output.

Syntax

```
public const char * log_file_name;
```

Remarks

If send is TRUE, the error log is sent to the consolidated (unless this pointer is NULL).

log_size a_sql_uint32

DBRemoteSQL renames and restarts the online transaction log when the size of the online transaction log is greater than this value.

Syntax

```
public a_sql_uint32 log_size;
```

Remarks

Corresponds to the dbremote -x option.

logrtn MSG_CALLBACK

Pointer to a function that prints the given message to a log file.

Syntax

```
public MSG_CALLBACK logrtn;
```

Remarks

These messages do not need to be seen by the user.

max_length a_sql_uint32

Set to the maximum length (in bytes) a message can have.

Syntax

```
public a_sql_uint32 max_length;
```

Remarks

This affects sending and receiving. The recommended value is 50000. Corresponds to the dbremote -l option.

memory a_sql_uint32

Set to the maximum size (in bytes) of memory buffers to use while building messages to send.

Syntax

```
public a_sql_uint32 memory;
```

Remarks

The recommended value is at least $2 * 1024 * 1024$. Corresponds to the dbremote -m option.

mirror_logs char *

Pointer to the name of the directory containing offline mirror transaction logs.

Syntax

```
public char * mirror_logs;
```

Remarks

Corresponds to the dbremote -ml option.

more a_bit_field

This should be set to TRUE.

Syntax

```
public a_bit_field more;
```

msgqueuerrn MSG_QUEUE_CALLBACK

Function called by DBRemoteSQL when it wants to sleep.

Syntax

```
public MSG_QUEUE_CALLBACK msgqueuerrn;
```

Remarks

The parameter specifies the sleep period in milliseconds. The function should return the following, as defined in dllapi.h.

- MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. This is usually the value you should return.
- MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible.

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

no_user_interaction a_bit_field

When set TRUE, no user interaction is requested.

Syntax

```
public a_bit_field no_user_interaction;
```

operations a_sql_uint32

This value is used when applying messages.

Syntax

```
public a_sql_uint32 operations;
```

Remarks

Commits are ignored until DBRemoteSQL has at least this number of operations(inserts, deletes, updates) that are uncommitted. Corresponds to the dbremote -g option.

patience_retry a_sql_uint32

Set this to the number of polls for incoming messages that DBRemoteSQL should wait before assuming that a message it is expecting is lost.

Syntax

```
public a_sql_uint32 patience_retry;
```

Remarks

For example, if patience_retry is 3 then DBRemoteSQL tries up to three times to receive the missing message. Afterward, it sends a resend request. The recommended value is 1. Corresponds to the dbremote -rp option.

progress_index rtn SET_PROGRESS_CALLBACK

Pointer to a function that updates the state of the progress bar.

Syntax

```
public SET_PROGRESS_CALLBACK progress_index_rtn;
```

Remarks

This function takes two unsigned integer arguments index and max. On the first call, the values are the minimum and maximum values (for example, 0, 100). On subsequent calls, the first argument is the current index value (for example, between 0 and 100) and the second argument is always 0.

progress_msg_rtn MSG_CALLBACK

Pointer to a function that displays a progress message.

Syntax

```
public MSG_CALLBACK progress_msg_rtn;
```

queueparms char *

Reserved for internal use and must set to NULL.

Syntax

```
public char * queueparms;
```

receive_a_bit_field

When set TRUE, messages are received.

Syntax

```
public a_bit_field receive;
```

Remarks

If receive and send are both FALSE then both are assumed TRUE. It is recommended to set receive and send FALSE. Corresponds to the dbremote -r option.

receive_delay a_sql_uint32

Set this to the time (in seconds) to wait between polls for new incoming messages.

Syntax

```
public a_sql_uint32 receive_delay;
```

Remarks

The recommended value is 60. Corresponds to the dbremote -rd option.

remote_output_file_name char *

Pointer to the name of the DBRemoteSQL remote output file.

Syntax

```
public char * remote_output_file_name;
```

Remarks

Corresponds to the dbremote -ro or -rt option.

rename_log a_bit_field

When set TRUE, logs are renamed and restarted (DBRemoteSQL only).

Syntax

```
public a_bit_field rename_log;
```

resend_urgency a_sql_uint32

Set the time (in seconds) that DBRemoteSQL waits after seeing that a user needs a rescan before performing a full scan of the log.

Syntax

```
public a_sql_uint32 resend_urgency;
```

Remarks

Set to zero to allow DBRemoteSQL to choose a good value based on user send times and other information it has collected. Corresponds to the dbremote -ru option.

scan_log a_bit_field

Reserved for internal use and must set to FALSE.

Syntax

```
public a_bit_field scan_log;
```

send a_bit_field

When set TRUE, messages are sent.

Syntax

```
public a_bit_field send;
```

Remarks

If receive and send are both FALSE then both are assumed TRUE. It is recommended to set receive and send FALSE. Corresponds to the dbremote -s option.

send_delay a_sql_uint32

Set the time (in seconds) between scans of the log file for new operations to send.

Syntax

```
public a_sql_uint32 send_delay;
```

Remarks

Set to zero to allow DBRemoteSQL to choose a good value based on user send times. Corresponds to the dbremote -sd option.

set_window_title_rtn SET_WINDOW_TITLE_CALLBACK

Pointer to a function that resets the title of the window (Windows only).

Syntax

```
public SET_WINDOW_TITLE_CALLBACK set_window_title_rtn;
```

Remarks

The title could be "database_name (receiving, scanning, or sending) - default_window_title".

threads a_sql_uint32

Set the number of worker threads that should be used to apply messages.

Syntax

```
public a_sql_uint32 threads;
```

Remarks

This value must not exceed 50. Corresponds to the dbremote -w option.

transaction_logs char *

Should identify the directory with offline transaction logs (DBRemoteSQL only).

Syntax

```
public char * transaction_logs;
```

Remarks

Corresponds to the transaction_logs_directory argument of dbremote.

triggers a_bit_field

This should usually be cleared (FALSE) in most cases.

Syntax

```
public a_bit_field triggers;
```

Remarks

When set TRUE, trigger actions are replicated. Care should be exercised.

truncate_remote_output_file a_bit_field

When set TRUE, the remote output file is truncated rather than appended to.

Syntax

```
public a_bit_field truncate_remote_output_file;
```

Remarks

Corresponds to the dbremote -rt option.

unused a bit field

Reserved for internal use and must set to FALSE.

Syntax

```
public a_bit_field unused;
```

use_hex_offsets a bit field

When set TRUE, log offsets are shown in hexadecimal notation; otherwise decimal notation is used.

Syntax

```
public a_bit_field use_hex_offsets;
```

use_relative_offsets a bit field

When set TRUE, log offsets are displayed as relative to the start of the current log file.

Syntax

```
public a_bit_field use_relative_offsets;
```

Remarks

When set FALSE, log offsets from the beginning of time are displayed.

verbose a bit field

When set, extra information is produced.

Syntax

```
public a_bit_field verbose;
```

Remarks

Corresponds to the dbremote -v option.

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

warningrtn MSG_CALLBACK

Pointer to a function that displays the given warning message.

Syntax

```
public MSG_CALLBACK warningrtn;
```

Remarks

If NULL, the errorrtn function is called instead.

a_sync_db structure

Holds information needed for the dbmlsync utility using the DBTools library.

Syntax

```
typedef struct a_sync_db
```

Remarks

Some members correspond to features accessible from the dbmlsync command line utility. Unused members should be assigned the value 0, FALSE, or NULL, depending on data type.

allow_outside_connect a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field allow_outside_connect;
```

allow_schema_change a_bit_field

Set TRUE to check for schema changes between synchronizations.

Syntax

```
public a_bit_field allow_schema_change;
```

Remarks

Equivalent to the dbmlsync -sc option.

apply_dnld_file const char *

Name of download file to apply.

Syntax

```
public const char * apply_dnld_file;
```

Remarks

Equivalent to dbmlsync -ba option or NULL if option not specified.

argv char **

The argv array for this run, the last element of the array must be NULL.

Syntax

```
public char ** argv;
```

autoclose a_bit_field

Set TRUE to close window on completion.

Syntax

```
public a_bit_field autoclose;
```

Remarks

Equivalent to the dbmsync -qc option.

background_retry a_sql_int32

Number of times to retry an interrupted background synchronization.

Syntax

```
public a_sql_int32 background_retry;
```

Remarks

Equivalent to the dbmsync -bkr option.

background_sync a_bit_field

Set TRUE to do a background synchronization.

Syntax

```
public a_bit_field background_sync;
```

Remarks

Equivalent to the dbmsync -bk option.

cache_verbosity a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field cache_verbosity;
```

ce_argv char **

Reserved; use NULL.

Syntax

```
public char ** ce_argv;
```

ce_reproc_argv char **

Reserved; use NULL.

Syntax

```
public char ** ce_reproc_argv;
```

changing_pwd a_bit_field

Set TRUE when setting a new MobiLink password.

Syntax

```
public a_bit_field changing_pwd;
```

Remarks

See new_mlpassword field. Equivalent to the dbmlsync -mn option.

confirmrtn MSG_CALLBACK

Address of a confirmation request callback routine or NULL.

Syntax

```
public MSG_CALLBACK confirmrtn;
```

connectparms char *

Parameters needed to connect to the database.

Syntax

```
public char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:

```
"UID=DBA;PWD=sql;DBF=demo.db".
```

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```

connectparms_allocated a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field connectparms_allocated;
```

continue_download a_bit_field

Set TRUE to continue a previously failed download.

Syntax

```
public a_bit_field continue_download;
```

Remarks

Equivalent to the dbmlsync -dc option.

create_dnld_file const char *

Name of download file to create.

Syntax

```
public const char * create_dnld_file;
```

Remarks

Equivalent to dbmlsync -bc option or NULL if option not specified.

debug a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field debug;
```

debug_dump_char a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field debug_dump_char;
```

debug_dump_hex a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field debug_dump_hex;
```


debug_dump_size a_sql_uint32

Reserved; use 0.

Syntax

```
public a_sql_uint32 debug_dump_size;
```

debug_page_offsets a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field debug_page_offsets;
```

default_window_title char *

Name of the program to display in the window caption (for example, DBMLSync).

Syntax

```
public char * default_window_title;
```

dl_insert_width a_sql_uint32

Reserved; use 0.

Syntax

```
public a_sql_uint32 dl_insert_width;
```

dl_use_put a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field dl_use_put;
```

dlg_info_msg a_sql_uint32

Reserved; use 0.

Syntax

```
public a_sql_uint32 dlg_info_msg;
```

dnld_fail_len a_sql_uint32

Reserved; use 0.

Syntax

```
public a_sql_uint32 dnld_fail_len;
```

dnld_file_extra const char *

Specify extra string to include in download file.

Syntax

```
public const char * dnld_file_extra;
```

Remarks

Equivalent to dbmsync -be option.

dnld_gen_num a_bit_field

Set TRUE to update generation number when download file is applied.

Syntax

```
public a_bit_field dnld_gen_num;
```

Remarks

Equivalent to the dbmsync -bg option.

dnld_read_size a_sql_uint32

Set the download read size.

Syntax

```
public a_sql_uint32 dnld_read_size;
```

Remarks

Equivalent to the dbmsync -drs option.

download_only a_bit_field

Set TRUE to perform download-only synchronization.

Syntax

```
public a_bit_field download_only;
```

Remarks

Equivalent to the dbmsync -ds option.

encrypted_stream_opts const char *

Reserved; use NULL.

Syntax

```
public const char * encrypted_stream_opts;
```

encryption_key char *

The encryption key for the database file.

Syntax

```
public char * encryption_key;
```

Remarks

Equivalent to the dbmsync -ek option.

entered_dialog a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field entered_dialog;
```

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

est_upld_row_cnt a_sql_uint32

Set the estimated upload row count (for optimization).

Syntax

```
public a_sql_uint32 est_upld_row_cnt;
```

Remarks

Equivalent to the dbmsync -urc option.

extended_options char *

Extended options in the form "keyword=value;...".

Syntax

```
public char * extended_options;
```

Remarks

Equivalent to dbmsync -e option.

hide_conn_str a_bit_field

Set FALSE to show connect string, TRUE to hide the connect string.

Syntax

```
public a_bit_field hide_conn_str;
```

Remarks

Equivalent to the dbmlsync -vc option.

hide_ml_pwd a_bit_field

Set FALSE to show MobiLink password, TRUE to hide the MobiLink password.

Syntax

```
public a_bit_field hide_ml_pwd;
```

Remarks

Equivalent to the dbmlsync -vp option.

hovering_frequency a_sql_uint32

Set the logscan polling period in seconds.

Syntax

```
public a_sql_uint32 hovering_frequency;
```

Remarks

Usually 60. Equivalent to the dbmlsync -pp option.

ignore_debug_interrupt a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field ignore_debug_interrupt;
```

ignore_hook_errors a_bit_field

Set TRUE to ignore errors that occur in hook functions.

Syntax

```
public a_bit_field ignore_hook_errors;
```

Remarks

Equivalent to the dbmlsync -eh option.

ignore_hovering a_bit_field

Set TRUE to disable logscan polling.

Syntax

```
public a_bit_field ignore_hovering;
```

Remarks

Equivalent to the dbmsync -p option.

ignore_scheduling a_bit_field

Set TRUE to ignore scheduling.

Syntax

```
public a_bit_field ignore_scheduling;
```

Remarks

Equivalent to the dbmsync -is option.

include_scan_range const char *

Reserved; use NULL.

Syntax

```
public const char * include_scan_range;
```

init_cache a_sql_uint32

Initial size for cache.

Syntax

```
public a_sql_uint32 init_cache;
```

Remarks

Equivalent to the dbmsync -ci option.

init_cache_suffix char

Suffix for initial cache size ('B' for bytes, 'P' for percentage, or 0 if not specified).

Syntax

```
public char init_cache_suffix;
```

kill_other_connections a_bit_field

Set TRUE to drop connections with locks on tables being synchronized.

Syntax

```
public a_bit_field kill_other_connections;
```

Remarks

Equivalent to the dbmsync -d option.

last_upload_def a_syncpub *

Reserved; use NULL.

Syntax

```
public a_syncpub * last_upload_def;
```

lite_blob_handling a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field lite_blob_handling;
```

log_file_name const char *

Database server message log file name.

Syntax

```
public const char * log_file_name;
```

Remarks

Equivalent to dbmsync -o or -ot option.

log_size a_sql_uint32

Size in bytes of log file when renaming and restarting the transaction log.

Syntax

```
public a_sql_uint32 log_size;
```

Remarks

Specify 0 for unspecified size. Equivalent to the dbmsync -x option.

logrtn MSG_CALLBACK

Address of a logging callback routine to write messages only to a log file or NULL.

Syntax

```
public MSG_CALLBACK logrtn;
```

max_cache a_sql_uint32

Maximum size for cache.

Syntax

```
public a_sql_uint32 max_cache;
```

Remarks

Equivalent to the dbmsync -cm option.

max_cache_suffix char

Suffix for maximum cache size ('B' for bytes, 'P' for percentage, or 0 if not specified).

Syntax

```
public char max_cache_suffix;
```

min_cache a_sql_uint32

Minimum size for cache.

Syntax

```
public a_sql_uint32 min_cache;
```

Remarks

Equivalent to the dbmsync -cl option.

min_cache_suffix char

Suffix for minimum cache size ('B' for bytes, 'P' for percentage, or 0 if not specified).

Syntax

```
public char min_cache_suffix;
```

mlpassword char *

The MobiLink password or NULL, if the option is not specified.

Syntax

```
public char * mlpassword;
```

Remarks

Equivalent to the dbmlsync -mp option.

msgqueue rtn MSG_QUEUE_CALLBACK

Function called by DBMLSync when it wants to sleep.

Syntax

```
public MSG_QUEUE_CALLBACK msgqueue rtn;
```

Remarks

The parameter specifies the sleep period in milliseconds. The function should return the following, as defined in dllapi.h.

- MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. This is usually the value you should return.
- MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible.
- MSGQ_SYNC_REQUESTED indicates that the routine slept for less than the requested number of milliseconds and that the next synchronization should begin immediately if a synchronization is not currently in progress.

msg rtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msg rtn;
```

new_m lpassword char *

The new MobiLink password or NULL, if the option is not specified.

Syntax

```
public char * new_m lpassword;
```

Remarks

Equivalent to the dbmlsync -mn option.

no_offline_logscan a_sql_uint32

Set TRUE to disable offline logscan (cannot use with -x).

Syntax

```
public a_sql_uint32 no_offline_logscan;
```

Remarks

Equivalent to the dbmlsync -do option.

no_schema_cache a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field no_schema_cache;
```

no_stream_compress a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field no_stream_compress;
```

offline_dir const char *

Transaction logs directory.

Syntax

```
public const char * offline_dir;
```

Remarks

Last item specified on dbmsync command line.

output_to_file a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field output_to_file;
```

output_to_mobile_link a_bit_field

Reserved; use 1.

Syntax

```
public a_bit_field output_to_mobile_link;
```

persist_connection a_bit_field

Set TRUE to persist the MobiLink connection between synchronizations.

Syntax

```
public a_bit_field persist_connection;
```

Remarks

Set FALSE to close the MobiLink connection between synchronizations. Equivalent to the dbmsync -pc{+|-} option.

ping a bit field

Set TRUE to ping MobiLink server.

Syntax

```
public a_bit_field ping;
```

Remarks

Equivalent to the dbmlsync -pi option.

preload dlls char *

Reserved; use NULL.

Syntax

```
public char * preload_dlls;
```

progress_index_rtn SET_PROGRESS_CALLBACK

Function called to update the state of the progress bar.

Syntax

```
public SET_PROGRESS_CALLBACK progress_index_rtn;
```

progress_msg_rtn MSG_CALLBACK

Function called to change the text in the status window, above the progress bar.

Syntax

```
public MSG_CALLBACK progress_msg_rtn;
```

prompt_again a bit field

Reserved; use 0.

Syntax

```
public a_bit_field prompt_again;
```

prompt_for_encrypt_key a bit field

Reserved; use 0.

Syntax

```
public a_bit_field prompt_for_encrypt_key;
```

protocol_add_cli_bit_to_cli_both a bit field

Reserved; use 0.

Syntax

```
public a_bit_field protocol_add_cli_bit_to_cli_both;
```

protocol_add_cli_bit_to_cli_max a bit field

Reserved; use 0.

Syntax

```
public a_bit_field protocol_add_cli_bit_to_cli_max;
```

protocol_add_serv_bit_to_cli_both a bit field

Reserved; use 0.

Syntax

```
public a_bit_field protocol_add_serv_bit_to_cli_both;
```

protocol_add_serv_bit_to_cli_max a bit field

Reserved; use 0.

Syntax

```
public a_bit_field protocol_add_serv_bit_to_cli_max;
```

protocol_add_serv_bit_to_serv_both a bit field

Reserved; use 0.

Syntax

```
public a_bit_field protocol_add_serv_bit_to_serv_both;
```

protocol_add_serv_bit_to_serv_max a bit field

Reserved; use 0.

Syntax

```
public a_bit_field protocol_add_serv_bit_to_serv_max;
```

raw_file const char *

Reserved; use NULL.

Syntax

```
public const char * raw_file;
```

rename_log a_bit_field

Set TRUE to rename and restart the transaction log.

Syntax

```
public a_bit_field rename_log;
```

Remarks

See log_size field. Equivalent to the dbmlsync -x option.

reserved a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field reserved;
```

retry_remote_ahead a_bit_field

Set TRUE to resend upload using remote offset on progress mismatch when remote offset is greater than consolidated offset.

Syntax

```
public a_bit_field retry_remote_ahead;
```

Remarks

Equivalent to the dbmlsync -ra option.

retry_remote_behind a_bit_field

Set TRUE to resend upload using remote offset on progress mismatch.

Syntax

```
public a_bit_field retry_remote_behind;
```

Remarks

when remote offset is less than consolidated offset. Equivalent to the dbmlsync -r or -rb option.

server_mode a_bit_field

Set TRUE to run in server mode.

Syntax

```
public a_bit_field server_mode;
```

Remarks

Equivalent to the dbmlsync -sm option.

server_port a_sql_uint32

Set communication port when running in server mode.

Syntax

```
public a_sql_uint32 server_port;
```

Remarks

Equivalent to the dbmsync -po option.

set_window_title_rtn SET_WINDOW_TITLE_CALLBACK

Function to call to change the title of the dbmsync window (Windows only).

Syntax

```
public SET_WINDOW_TITLE_CALLBACK set_window_title_rtn;
```

status_rtn STATUS_CALLBACK

Reserved; use NULL.

Syntax

```
public STATUS_CALLBACK status_rtn;
```

strictly_free_memory a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field strictly_free_memory;
```

strictly_ignore_trigger_ops a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field strictly_ignore_trigger_ops;
```

sync_opt char *

Reserved; use NULL.

Syntax

```
public char * sync_opt;
```

sync_params char *

User authentication parameters.

Syntax

```
public char * sync_params;
```

Remarks

Equivalent to the dbmsync -ap option.

sync_profile char *

Synchronization profile to execute.

Syntax

```
public char * sync_profile;
```

Remarks

Equivalent to the dbmsync -sp option.

trans_upload a_bit_field

Set TRUE to upload each database transaction separately.

Syntax

```
public a_bit_field trans_upload;
```

Remarks

Equivalent to the dbmsync -tu option.

upld_fail_len a_sql_uint32

Reserved; use 0.

Syntax

```
public a_sql_uint32 upld_fail_len;
```

upload_defs a_syncpub *

Linked list of publications/subscriptions to synchronize.

Syntax

```
public a_syncpub * upload_defs;
```

upload_only a bit field

Set TRUE to perform upload-only synchronization.

Syntax

```
public a_bit_field upload_only;
```

Remarks

Equivalent to the dbmlsync -uo option.

usage_rtn USAGE_CALLBACK

Reserved; use NULL.

Syntax

```
public USAGE_CALLBACK usage_rtn;
```

use_fixed_cache a bit field

Reserved; use 0.

Syntax

```
public a_bit_field use_fixed_cache;
```

use_hex_offsets a bit field

Reserved; use 0.

Syntax

```
public a_bit_field use_hex_offsets;
```

use_relative_offsets a bit field

Reserved; use 0.

Syntax

```
public a_bit_field use_relative_offsets;
```

used_dialog_allocation a bit field

Reserved; use 0.

Syntax

```
public a_bit_field used_dialog_allocation;
```

user_name char *

The MobiLink user to synchronize (deprecated).

Syntax

```
public char * user_name;
```

Remarks

Equivalent to the dbmlsync -u option.

verbose a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field verbose;
```

verbose_download a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field verbose_download;
```

verbose_download_data a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field verbose_download_data;
```

verbose_hook a_bit_field

Set TRUE to show hook script information.

Syntax

```
public a_bit_field verbose_hook;
```

Remarks

Equivalent to the dbmlsync -vs option.

verbose_minimum a_bit_field

Set TRUE to set verbosity at a minimum.

Syntax

```
public a_bit_field verbose_minimum;
```


Remarks

Equivalent to the dbmlsync -v option.

verbose_msgid a_bit_field

Set TRUE to show message IDs.

Syntax

```
public a_bit_field verbose_msgid;
```

Remarks

Equivalent to the dbmlsync -vi option.

verbose_option_info a_bit_field

Set TRUE to show command line and extended options.

Syntax

```
public a_bit_field verbose_option_info;
```

Remarks

Equivalent to the dbmlsync -vo option.

verbose_protocol a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field verbose_protocol;
```

verbose_row_cnts a_bit_field

Set TRUE to show upload/download row counts.

Syntax

```
public a_bit_field verbose_row_cnts;
```

Remarks

Equivalent to the dbmlsync -vn option.

verbose_row_data a_bit_field

Set TRUE to show upload/download row values.

Syntax

```
public a_bit_field verbose_row_data;
```

Remarks

Equivalent to the dbmlsync -vr option.

verbose_server a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field verbose_server;
```

verbose_upload a_bit_field

Set TRUE to show upload stream information.

Syntax

```
public a_bit_field verbose_upload;
```

Remarks

Equivalent to the dbmlsync -vu option.

verbose_upload_data a_bit_field

Reserved; use 0.

Syntax

```
public a_bit_field verbose_upload_data;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

warningrtn MSG_CALLBACK

Function called to display warning messages.

Syntax

```
public MSG_CALLBACK warningrtn;
```

a_syncpub structure

Holds information needed for the dbmlsync utility.

Syntax

```
typedef struct a_syncpub
```

ext_opt char *

Extended options in the form "keyword=value;...".

Syntax

```
public char * ext_opt;
```

Remarks

These are the same options the would follow the dbmlsync -eu option.

next struct a_syncpub *

Pointer to the next node in the list, NULL for the last node.

Syntax

```
public struct a_syncpub * next;
```

pub_name char *

Publication name(s) separated by commas (deprecated).

Syntax

```
public char * pub_name;
```

Remarks

This is the same string that would follow the dbmlsync -n option. Only 1 of pub_name and subscription may be non-NULL.

subscription char *

Subscription name(s) separated by commas.

Syntax

```
public char * subscription;
```

Remarks

This is the same string the would follow the dbmlsync -s option. Only 1 of pub_name and subscription may be non-NULL.

a_sysinfo structure

Holds information needed for dbinfo and dbunload utilities using the DBTools library.

Syntax

```
typedef struct a_sysinfo
```

blank_padding a_bit_field

1 if blank padding is used in this database, 0 otherwise.

Syntax

```
public a_bit_field blank_padding;
```

case_sensitivity a_bit_field

1 if the database is case sensitive, 0 otherwise.

Syntax

```
public a_bit_field case_sensitivity;
```

default_collation char

The collation sequence for the database.

Syntax

```
public char default_collation;
```

encryption a_bit_field

1 if the database is encrypted, 0 otherwise.

Syntax

```
public a_bit_field encryption;
```

page_size unsigned short

The page size for the database.

Syntax

```
public unsigned short page_size;
```

valid_data a_bit_field

1 to indicate that the other bit fields are valid.

Syntax

```
public a_bit_field valid_data;
```

a_table_info structure

Holds information about a table needed as part of the a_db_info structure.

Syntax

```
typedef struct a_table_info
```

index_pages a_sql_uint32

Number of index pages.

Syntax

```
public a_sql_uint32 index_pages;
```

index_used a_sql_uint32

Number of bytes used in index pages.

Syntax

```
public a_sql_uint32 index_used;
```

index_used_pct a_sql_uint32

Index space utilization as a percentage.

Syntax

```
public a_sql_uint32 index_used_pct;
```

next struct a_table_info *

Next table in the list.

Syntax

```
public struct a_table_info * next;
```

table_id a_sql_uint32

ID number for this table.

Syntax

```
public a_sql_uint32 table_id;
```

table_name char *

Name of the table.

Syntax

```
public char * table_name;
```

table_pages a_sql_uint32

Number of table pages.

Syntax

```
public a_sql_uint32 table_pages;
```

table_used a_sql_uint32

Number of bytes used in table pages.

Syntax

```
public a_sql_uint32 table_used;
```

table_used_pct a_sql_uint32

Table space utilization as a percentage.

Syntax

```
public a_sql_uint32 table_used_pct;
```

a_translate_log structure

Holds information needed for transaction log translation using the DBTools library.

Syntax

```
typedef struct a_translate_log
```

ansi_sql a_bit_field

Set TRUE to produce ANSI standard SQL transactions.

Syntax

```
public a_bit_field ansi_sql;
```

Remarks

Set TRUE by dbtran -s option.

chronological_order a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field chronological_order;
```

comment_trigger_trans a_bit_field

Set TRUE to include trigger-generated transactions as comments.

Syntax

```
public a_bit_field comment_trigger_trans;
```

Remarks

Set TRUE by dbtran -z option.

confirmrtn MSG_CALLBACK

Address of a confirmation request callback routine or NULL.

Syntax

```
public MSG_CALLBACK confirmrtn;
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:

```
"UID=DBA;PWD=sql;DBF=demo.db".
```

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```

debug a bit field

Reserved; set to FALSE.

Syntax

```
public a_bit_field debug;
```

debug_dump char a bit field

Reserved; set to FALSE.

Syntax

```
public a_bit_field debug_dump_char;
```

debug_dump_hex a bit field

Reserved; set to FALSE.

Syntax

```
public a_bit_field debug_dump_hex;
```

debug_dump_size a_sql_uint32

Reserved, use 0.

Syntax

```
public a_sql_uint32 debug_dump_size;
```

debug_page_offsets a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field debug_page_offsets;
```

debug_sql_remote a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field debug_sql_remote;
```

encryption_key const char *

The encryption key for the database file. Equivalent to dbtran -ek option.

Syntax

```
public const char * encryption_key;
```

errortn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errortn;
```

extra_audit a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field extra_audit;
```

force_chaining a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field force_chaining;
```


force_recovery a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field force_recovery;
```

generate_reciprocals a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field generate_reciprocals;
```

include_audit a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field include_audit;
```

include_destination_sets const char *

Reserved, use NULL.

Syntax

```
public const char * include_destination_sets;
```

include_publications const char *

Reserved, use NULL.

Syntax

```
public const char * include_publications;
```

include_scan_range const char *

Reserved, use NULL.

Syntax

```
public const char * include_scan_range;
```

include_source_sets const char *

Reserved, use NULL.

Syntax

```
public const char * include_source_sets;
```

include_subsets a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field include_subsets;
```

include_tables const char *

Reserved, use NULL.

Syntax

```
public const char * include_tables;
```

include_trigger_trans a_bit_field

Set TRUE to include trigger-generated transactions.

Syntax

```
public a_bit_field include_trigger_trans;
```

Remarks

Set TRUE by dbtran -t, -g and -sr options.

leave_output_on_error a_bit_field

Set TRUE to leave the generated SQL file if log error detected.

Syntax

```
public a_bit_field leave_output_on_error;
```

Remarks

Set TRUE by dbtran -k option.

logname const char *

Name of the transaction log file. If NULL, there is no log.

Syntax

```
public const char * logname;
```

logrtn MSG_CALLBACK

Address of a logging callback routine to write messages only to a log file or NULL.

Syntax

```
public MSG_CALLBACK logrtn;
```

logs_dir const char *

Transaction logs directory.

Syntax

```
public const char * logs_dir;
```

Remarks

Equivalent to dbtran -m option. The sqlname pointer must be set and connectparms must be NULL.

match_mode a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field match_mode;
```

match_pos const char *

Reserved, use NULL.

Syntax

```
public const char * match_pos;
```

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

omit_comments a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field omit_comments;
```

queueparms const char *

Reserved, use NULL.

Syntax

```
public const char * queueparms;
```

quiet a_bit_field

Set to TRUE to operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dbtran -q option.

recovery_bytes a_sql_uint32

Reserved, use 0.

Syntax

```
public a_sql_uint32 recovery_bytes;
```

recovery_ops a_sql_uint32

Reserved, use 0.

Syntax

```
public a_sql_uint32 recovery_ops;
```

remove_rollback a_bit_field

Set to FALSE if you want to include rollback transactions in output.

Syntax

```
public a_bit_field remove_rollback;
```

Remarks

Set FALSE by dbtran -a option.

replace a_bit_field

Set TRUE to replace the SQL file without a confirmation.

Syntax

```
public a_bit_field replace;
```

Remarks

Set TRUE by dbtran -y option.

repsrver_users const char *

Reserved, use NULL.

Syntax

```
public const char * repsrver_users;
```

show_undo a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field show_undo;
```

since_checkpoint a_bit_field

Set TRUE for output from most recent checkpoint.

Syntax

```
public a_bit_field since_checkpoint;
```

Remarks

Set TRUE by dbtran -f option.

since_time a_sql_uint32

Output from most recent checkpoint before time.

Syntax

```
public a_sql_uint32 since_time;
```

Remarks

The number of minutes since January 1, 0001. Equivalent to dbtran -j option.

sqlname const char *

Name of the SQL output file.

Syntax

```
public const char * sqlname;
```

Remarks

If NULL, then the name is based on the transaction log file name. Equivalent to dbtran -n option.

statusrtn MSG_CALLBACK

Address of a status message callback routine or NULL.

Syntax

```
public MSG_CALLBACK statusrtn;
```

use_hex_offsets a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field use_hex_offsets;
```

use_relative_offsets a_bit_field

Reserved; set to FALSE.

Syntax

```
public a_bit_field use_relative_offsets;
```

userlist p_name

A linked list of user names.

Syntax

```
public p_name userlist;
```

Remarks

Equivalent to dbtran -u user1,... or -x user1,... Select or omit transactions for listed users.

userlisttype char

Set to DBTRAN_INCLUDE_ALL unless you want to include or exclude a list of users.

Syntax

```
public char userlisttype;
```

Remarks

DBTRAN_INCLUDE_SOME for -u, or DBTRAN_EXCLUDE_SOME for -x.

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_truncate_log structure

Holds information needed for transaction log truncation using the DBTools library.

Syntax

```
typedef struct a_truncate_log
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:

```
"UID=DBA;PWD=sql;DBF=demo.db".
```

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

quiet a_bit_field

Set TRUE to operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dbbackup -q option.

server_backup a_bit_field

Set TRUE to indicate backup on server using BACKUP DATABASE.

Syntax

```
public a_bit_field server_backup;
```

Remarks

Set TRUE by dbbackup -s option when dbbackup -x option is specified.

truncate_interrupted char

Truncate was interrupted if non-zero.

Syntax

```
public char truncate_interrupted;
```

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

a_validate_db structure

Holds information needed for database validation using the DBTools library.

Syntax

```
typedef struct a_validate_db
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:

```
"UID=DBA;PWD=sql;DBF=demo.db".
```

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```


errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

index a_bit_field

Set TRUE to validate indexes.

Syntax

```
public a_bit_field index;
```

Remarks

The tables field points to a list of indexes. Set TRUE by dbvalid -i option. Set FALSE by dbvalid -t option.

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

quiet a_bit_field

Set TRUE to operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dbvalid -q option.

statusrtn MSG_CALLBACK

Address of a status message callback routine or NULL.

Syntax

```
public MSG_CALLBACK statusrtn;
```

tables p_name

Pointer to a linked list of table names or index names (when the index field is set TRUE).

Syntax

```
public p_name tables;
```

Remarks

This is set by the dbvalid object-name-list argument.

type char

The type of validation to perform.

Syntax

```
public char type;
```

Remarks

One of VALIDATE_NORMAL, VALIDATE_EXPRESS, VALIDATE_CHECKSUM, etc.
See Validation enumeration.

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

an_erase_db structure

Holds information needed to erase a database using the DBTools library.

Syntax

```
typedef struct an_erase_db
```

confirmrtn MSG_CALLBACK

Address of a confirmation request callback routine or NULL.

Syntax

```
public MSG_CALLBACK confirmrtn;
```

dbname const char *

Database file name.

Syntax

```
public const char * dbname;
```

encryption_key const char *

The encryption key for the database file.

Syntax

```
public const char * encryption_key;
```

Remarks

Equivalent to dberase -ek or -ep options.

erase a bit field

Erase without confirmation (1) or with confirmation (0).

Syntax

```
public a_bit_field erase;
```

Remarks

Set TRUE by dberase -y option.

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

msggrtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msggrtn;
```

quiet a bit field

Operate without printing messages (1), or print messages (0).

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dberase -q option.

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

an_unload_db structure

Holds information needed to unload a database using the DBTools library or extract a remote database for SQL Remote.

Syntax

```
typedef struct an_unload_db
```

Remarks

Those fields used by the dbextract SQL Remote Extraction utility are indicated.

compress_output a bit field

Set TRUE to compress table data files.

Syntax

```
public a_bit_field compress_output;
```

Remarks

Set TRUE by dbunload -cp option.

confirmrtn MSG_CALLBACK

Address of a confirmation request callback routine or NULL.

Syntax

```
public MSG_CALLBACK confirmrtn;
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:
"UID=DBA;PWD=sql;DBF=demo.db".

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```

debug a bit field

Reserved; set FALSE.

Syntax

```
public a_bit_field debug;
```

display_create a bit field

Set TRUE to display database creation command (sql or dbinit).

Syntax

```
public a_bit_field display_create;
```

Remarks

Set TRUE by dbunload -cm sql or -cm dbinit option.

display_create dbinit a bit field

Set TRUE to display dbinit database creation command.

Syntax

```
public a_bit_field display_create_dbinit;
```

Remarks

Set TRUE by dbunload -cm dbinit option.

encrypted_tables a bit field

Set TRUE to enable encrypted tables in new database (with -an or -ar).

Syntax

```
public a_bit_field encrypted_tables;
```

Remarks

Set TRUE by dbunload/dbxtract -et option.

encryption_algorithm const char *

The encryption algorithm which may be "simple", "aes", "aes256", "aes_fips", "aes256_fips", or NULL for none.

Syntax

```
public const char * encryption_algorithm;
```

Remarks

Set by dbunload/dbxtract -ea option.

encryption_key const char *

The encryption key for the database file.

Syntax

```
public const char * encryption_key;
```

Remarks

Set by dbunload/dbxtract -ek or -ep option.

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

escape_char char

The escape character (normally, "\").

Syntax

```
public char escape_char;
```

Remarks

Used when escape_char_present is TRUE. Set TRUE by dbunload/dbxtract -p option.

escape_char_present a bit field

Set TRUE to indicate that the escape character in escape_char is defined.

Syntax

```
public a_bit_field escape_char_present;
```

Remarks

Set TRUE by dbunload/dbxtract -p option.

exclude_foreign_keys a bit field

Set TRUE to exclude foreign keys.

Syntax

```
public a_bit_field exclude_foreign_keys;
```

Remarks

Set TRUE by dbxtract -xf option.

exclude_hooks a bit field

Set TRUE to exclude procedure hooks.

Syntax

```
public a_bit_field exclude_hooks;
```

Remarks

Set TRUE by dbextract -xh option.

exclude_procedures a bit field

Set TRUE to exclude stored procedures.

Syntax

```
public a_bit_field exclude_procedures;
```

Remarks

Set TRUE by dbextract -xp option.

exclude_tables a bit field

Set FALSE to indicate that the list contains tables to be included.

Syntax

```
public a_bit_field exclude_tables;
```

Remarks

Set TRUE to indicate that the list contains tables to be excluded. Set TRUE by dbunload -e option.

exclude_triggers a bit field

Set TRUE to exclude triggers.

Syntax

```
public a_bit_field exclude_triggers;
```

Remarks

Set TRUE by dbextract -xt option.

exclude_views a bit field

Set TRUE to exclude views.

Syntax

```
public a_bit_field exclude_views;
```

Remarks

Set TRUE by dbextract -xv option.

extract a bit field

Set TRUE if performing a remote database extraction.

Syntax

```
public a_bit_field extract;
```

Remarks

Set FALSE by dbunload. Set TRUE by dbextract.

genscript a bit field

Reserved; set FALSE.

Syntax

```
public a_bit_field genscript;
```

include where subscribe a bit field

Set TRUE to extract fully qualified publications.

Syntax

```
public a_bit_field include_where_subscribe;
```

Remarks

Set TRUE by dbextract -f option.

isolation level unsigned short

The isolation level at which to operate.

Syntax

```
public unsigned short isolation_level;
```

Remarks

Set by dbextract -l option.

isolation set a bit field

Set TRUE to indicate that isolation_level has been set for all extraction operations.

Syntax

```
public a_bit_field isolation_set;
```


Remarks

Set TRUE by dbxtract -l option.

locale const char *

Reserved; use NULL.

Syntax

```
public const char * locale;
```

make_auxiliary a_bit_field

Set TRUE to make auxiliary catalog (for use with diagnostic tracing).

Syntax

```
public a_bit_field make_auxiliary;
```

Remarks

Set TRUE by dbunload -k option.

ms_filename const char *

Reserved; use NULL.

Syntax

```
public const char * ms_filename;
```

ms_reserve int

Reserved; use 0.

Syntax

```
public int ms_reserve;
```

ms_size int

Reserved; use 0.

Syntax

```
public int ms_size;
```

msg rtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msg rtn;
```

no_confirm a_bit_field

Set TRUE to replace an existing SQL script file without confirmation.

Syntax

```
public a_bit_field no_confirm;
```

Remarks

Set by dbunload/dbxtract -y option.

no_reload_status a_bit_field

Set TRUE to suppress reload status messages for tables and indexes.

Syntax

```
public a_bit_field no_reload_status;
```

Remarks

Set TRUE by dbunload -qr option.

notemp_size long

Reserved; use 0.

Syntax

```
public long notemp_size;
```

preserve_identity_values a_bit_field

Set TRUE to preserve identity values for AUTOINCREMENT columns.

Syntax

```
public a_bit_field preserve_identity_values;
```

Remarks

Set TRUE by dbunload -l option.

preserve_ids a_bit_field

Set TRUE to preserve user IDs.

Syntax

```
public a_bit_field preserve_ids;
```

Remarks

This is the normal setting. Set FALSE by dbunload -e option.

profiling_uses_single_dbspace a_bit_field

Set TRUE to collapse to a single dbspace file (for use with diagnostic tracing).

Syntax

```
public a_bit_field profiling_uses_single_dbspace;
```

Remarks

Set TRUE by dbunload -kd option.

recompute a_bit_field

Set TRUE to redo computed columns.

Syntax

```
public a_bit_field recompute;
```

Remarks

Set TRUE by dbunload -dc option.

refresh_mat_view a_bit_field

Set TRUE to generate statements to refresh text indexes and valid materialized views.

Syntax

```
public a_bit_field refresh_mat_view;
```

Remarks

Set TRUE by dbunload/dbxtract -g option.

reload_connectparms char *

Connection parameters such as user ID, password, and database for the reload database.

Syntax

```
public char * reload_connectparms;
```

Remarks

Set by dbunload/dbxtract -ac option.

reload_db_filename char *

Name of the new database file to create and reload.

Syntax

```
public char * reload_db_filename;
```

Remarks

Set by dbunload/dbxtract -an option.

reload_db_logname char *

Filename of the new database transaction log or NULL.

Syntax

```
public char * reload_db_logname;
```

Remarks

Set by dbxtract -al option.

reload_filename const char *

Name to use for the reload SQL script file (for example, reload.sql).

Syntax

```
public const char * reload_filename;
```

Remarks

Set by dbunload -r option.

reload_page_size unsigned short

The reloaded database page size.

Syntax

```
public unsigned short reload_page_size;
```

Remarks

Set by dbunload -ap option.

remote_dir const char *

Like temp_dir but for internal unloads on server side.

Syntax

```
public const char * remote_dir;
```

remove_encrypted_tables a_bit_field

Set TRUE to remove encryption from encrypted tables.

Syntax

```
public a_bit_field remove_encrypted_tables;
```

Remarks

Set TRUE by dbunload/dbxtract -er option.

replace_db a_bit_field

Set TRUE to replace the database.

Syntax

```
public a_bit_field replace_db;
```

Remarks

Set TRUE by dbunload -ar option.

runscript a_bit_field

Reserved; set FALSE.

Syntax

```
public a_bit_field runscript;
```

schema_reload a_bit_field

Reserved; set FALSE.

Syntax

```
public a_bit_field schema_reload;
```

site_name const char *

The site name to be used by dbxtract. NULL otherwise.

Syntax

```
public const char * site_name;
```

start_subscriptions a_bit_field

Set TRUE to start subscriptions.

Syntax

```
public a_bit_field start_subscriptions;
```

Remarks

This is the default for dbxtract. Set FALSE by dbxtract -b option.

startline const char *

Reserved; use NULL.

Syntax

```
public const char * startline;
```

startline_name a_bit_field

Reserved; set FALSE.

Syntax

```
public a_bit_field startline_name;
```

startline_old const char *

Reserved; use NULL.

Syntax

```
public const char * startline_old;
```

statusrtn MSG_CALLBACK

Address of a status message callback routine or NULL.

Syntax

```
public MSG_CALLBACK statusrtn;
```

subscriber_username const char *

The subscriber name to be used by dbxtract. NULL otherwise.

Syntax

```
public const char * subscriber_username;
```

suppress_statistics a_bit_field

Set TRUE to suppress inclusion of column statistics.

Syntax

```
public a_bit_field suppress_statistics;
```

Remarks

Set TRUE by dbunload -ss option.

sysinfo a_sysinfo

Reserved; use NULL.

Syntax

```
public a_sysinfo sysinfo;
```

table_list p_name

Selective table list.

Syntax

```
public p_name table_list;
```

Remarks

Set by dbunload -e and -t options.

table_list_provided a_bit_field

Set TRUE to indicate that a list of tables has been provided.

Syntax

```
public a_bit_field table_list_provided;
```

Remarks

See table_list field. Set TRUE by dbunload -e or -t options.

temp_dir const char *

Directory for unloading data files.

Syntax

```
public const char * temp_dir;
```

template_name const char *

The template name to be used by dbxtract. NULL otherwise.

Syntax

```
public const char * template_name;
```

unload_interrupted char

Reserved; set to 0.

Syntax

```
public char unload_interrupted;
```

unload_type char

Set Unload enumeration (UNLOAD_ALL and so on).

Syntax

```
public char unload_type;
```

Remarks

Set by dbunload/dbxtract -d, -k, -n options.

unordered a_bit_field

Set TRUE for unordered data.

Syntax

```
public a_bit_field unordered;
```

Remarks

Indexes will not be used to unload data. Set by dbunload/dbxtract -u option.

use_internal_reload a_bit_field

Set TRUE to perform an internal reload.

Syntax

```
public a_bit_field use_internal_reload;
```

Remarks

This is the normal setting. Set TRUE by dbunload/dbxtract -ii and -xi option. Set FALSE by dbunload/dbxtract -ix and -xx option.

use_internal_unload a_bit_field

Set TRUE to Perform an internal unload.

Syntax

```
public a_bit_field use_internal_unload;
```

Remarks

Set TRUE by dbunload/dbxtract -i? option. Set FALSE by dbunload/dbxtract -x? option.

verbose char

See Verbosity enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).

Syntax

```
public char verbose;
```


version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```

an_upgrade_db structure

Holds information needed to upgrade a database using the DBTools library.

Syntax

```
typedef struct an_upgrade_db
```

connectparms const char *

Parameters needed to connect to the database.

Syntax

```
public const char * connectparms;
```

Remarks

They take the form of connection strings, such as the following:

```
"UID=DBA;PWD=sql;DBF=demo.db".
```

The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny16\bin32\dbsrv16.exe".

A full example connection string including the START parameter:

```
"UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
```

errorrtn MSG_CALLBACK

Address of an error message callback routine or NULL.

Syntax

```
public MSG_CALLBACK errorrtn;
```

jconnect a_bit_field

Set TRUE to upgrade the database to include jConnect procedures.

Syntax

```
public a_bit_field jconnect;
```

Remarks

Set FALSE by dbupgrad -i option.

msg rtn MSG_CALLBACK

Address of an information message callback routine or NULL.

Syntax

```
public MSG_CALLBACK msg rtn;
```

quiet a bit field

Set TRUE to operate without printing messages.

Syntax

```
public a_bit_field quiet;
```

Remarks

Set TRUE by dbupgrad -q option.

restart a bit field

Set TRUE to restart the database after the upgrade.

Syntax

```
public a_bit_field restart;
```

Remarks

Set FALSE by the dbupgrad -nrs option.

status rtn MSG_CALLBACK

Address of a status message callback routine or NULL.

Syntax

```
public MSG_CALLBACK status rtn;
```

sys_proc_definer unsigned short

Assign 0 to upgrade the database to have the pre-16.0 SQL SECURITY model for legacy system stored procedures when upgrading from pre-16.0 releases.

Syntax

```
public unsigned short sys_proc_definer;
```

Remarks

When upgrading from a version 16.0 or later database retain the current SQL SECURITY model (same as not specifying -pd).

Assign 1 to upgrade the database to have the pre-16.0 SQL SECURITY model for legacy system stored procedures (same as -pd y)

Assign 2 to upgrade the database to have the pre-16.0 SQL SECURITY model for legacy system stored procedures (same as -pd n).

version unsigned short

DBTools version number (DB_TOOLS_VERSION_NUMBER).

Syntax

```
public unsigned short version;
```


Appendix: Using OLAP

OLAP (online analytical processing) is an efficient method of data analysis of information stored in a relational database.

Using OLAP you can analyze data on different dimensions, acquire result sets with subtotaled rows, and organize data into multidimensional cubes, all in a single SQL query. You can also use filters to drill down into the data, returning result sets quickly. This chapter describes the SQL/OLAP functionality that SAP Sybase IQ supports.

Note: The tables shown in OLAP examples are available in the `iqdemo` database.

About OLAP

The analytic functions, which offer the ability to perform complex data analysis within a single SQL statement, are facilitated by a category of software technology named online analytical processing (OLAP). Its functions are shown in the following list:

- **GROUP BY** clause extensions – **CUBE** and **ROLLUP**
- Analytical functions:
 - Simple aggregates – **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM**, **STDDEV** and **VARIANCE**

Note: You can use simple aggregate functions, except **Grouping()**, with an OLAP windowed function.

- Window functions:
 - Windowing aggregates – **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM**
 - Ranking functions – **RANK**, **DENSE_RANK**, **PERCENT_RANK**, and **NTILE**
 - Statistical functions – **STDDEV**, **STDDEV_SAMP**, **STDDEV_POP**, **VARIANCE**, **VAR_POP**, **VAR_SAMP**, **REGR_AVGX**, **REGR_AVGY**, **REGR_COUNT**, **REGR_INTERCEPT**, **REGR_R2**, **REGR_SLOPE**, **REGR_SXX**, **REGR_SXY**, **REGR_SYY**, **CORR**, **COVAR_POP**, **COVAR_SAMP**, **CUME_DIST**, **EXP_WEIGHTED_AVG**, and **WEIGHTED_AVG**.
 - Distribution functions – **PERCENTILE_CONT** and **PERCENTILE_DISC**
- Numeric functions – **WIDTH_BUCKET**, **CEIL**, and **LN**, **EXP**, **POWER**, **SQRT**, and **FLOOR**

Extensions to the ANSI SQL standard to include complex data analysis were introduced as an amendment to the 1999 SQL standard. SAP Sybase IQ SQL enhancements support these extensions.

Some database products provide a separate OLAP module that requires you to move data from the database into the OLAP module before analyzing it. By contrast, SAP Sybase IQ builds

OLAP features into the database itself, making deployment and integration with other database features, such as stored procedures, easy and seamless.

OLAP Benefits

OLAP functions, when combined with the **GROUPING**, **CUBE**, and **ROLLUP** extensions, provide two primary benefits.

First, they let you perform multidimensional data analysis, data mining, time series analyses, trend analysis, cost allocations, goal seeking, ad hoc multidimensional structural changes, nonprocedural modeling, and exception alerting, often with a single SQL statement. Second, the window and reporting aggregate functions use a relational operator, called a *window* that can be executed more efficiently than semantically equivalent queries that use self-joins or correlated subqueries. The result sets you obtain using OLAP can have subtotal rows and can be organized into multidimensional cubes.

Moving averages and moving sums can be calculated over various intervals; aggregations and ranks can be reset as selected column values change; and complex ratios can be expressed in simple terms. Within the scope of a single query expression, you can define several different OLAP functions, each with its own partitioning rules.

OLAP Evaluation

OLAP evaluation can be conceptualized as several phases of query execution that contribute to the final result.

You can identify OLAP phases of execution by the relevant clause in the query. For example, if a SQL query specification contains window functions, the **WHERE**, **JOIN**, **GROUP BY**, and **HAVING** clauses are processed first. Partitions are created after the groups defined in the **GROUP BY** clause and before the evaluation of the final **SELECT** list in the query's **ORDER BY** clause.

For the purpose of grouping, all NULL values are considered to be in the same group, even though NULL values are not equal to one another.

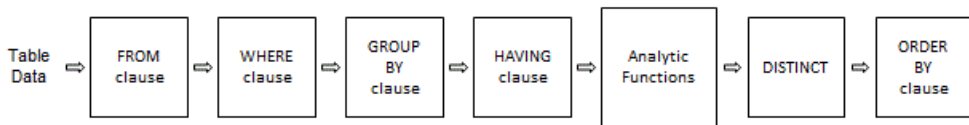
The **HAVING** clause acts as a filter, much like the **WHERE** clause, on the results of the **GROUP BY** clause.

Consider the semantics of a simple query specification involving the SQL statements and clauses, **SELECT**, **FROM**, **WHERE**, **GROUP BY**, and **HAVING** from the ANSI SQL standard:

1. The query produces a set of rows that satisfy the table expressions present in the **FROM** clause.
2. Predicates from the **WHERE** clause are applied to rows from the table. Rows that fail to satisfy the **WHERE** clause conditions (do not equal true) are rejected.
3. Except for aggregate functions, expressions from the **SELECT** list and in the list and **GROUP BY** clause are evaluated for every remaining row.

4. The resulting rows are grouped together based on distinct values of the expressions in the **GROUP BY** clause, treating NULL as a special value in each domain. The expressions in the **GROUP BY** clause serve as partition keys if a **PARTITION BY** clause is present.
5. For each partition, the aggregate functions present in the **SELECT** list or **HAVING** clause are evaluated. Once aggregated, individual table rows are no longer present in the intermediate result set. The new result set consists of the **GROUP BY** expressions and the values of the aggregate functions computed for each partition.
6. Conditions from the **HAVING** clause are applied to result groups. Groups are eliminated that do not satisfy the **HAVING** clause.
7. Results are partitioned on boundaries defined in the **PARTITION BY** clause. OLAP windows functions (rank and aggregates) are computed for result windows.

Figure 1: SQL processing for OLAP



GROUP BY Clause Extensions

Extensions to the **GROUP BY** clause let application developers write complex SQL statements that:

- Partition the input rows in multiple dimensions and combine multiple subsets of result groups.
- Create a “data cube,” providing a sparse, multi dimensional result set for data mining analyses.
- Create a result set that includes the original groups, and optionally includes a subtotal and grand-total row.

OLAP Grouping() operations, such as **ROLLUP** and **CUBE**, can be conceptualized as prefixes and subtotal rows.

Prefixes

A list of *prefixes* is constructed for any query that contains a **GROUP BY** clause. A prefix is a subset of the items in the **GROUP BY** clause and is constructed by excluding one or more of the rightmost items from those in the query’s **GROUP BY** clause. The remaining columns are called the *prefix columns*.

ROLLUP example 1—In the following **ROLLUP** example query, the **GROUP BY** list includes two variables, *Year* and *Quarter*:

```

SELECT year (OrderDate) AS Year, quarter(OrderDate)
       AS Quarter, COUNT(*) Orders
FROM SalesOrders
  
```

```
GROUP BY ROLLUP (Year, Quarter)
ORDER BY Year, Quarter
```

The query's two prefixes are:

- Exclude Quarter – the set of prefix columns contains the single column Year.
- Exclude both Quarter and Year – there are no prefix columns.

	Year	Quarter	Orders
Exclude Quarter and Year prefix	(NULL)	(NULL)	648
	2000	(NULL)	380
Exclude Quarter prefix	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
	2001	(NULL)	268
	2001	1	139
	2001	2	119
	2001	3	10

Note: The **GROUP BY** list contains the same number of prefixes as items.

Group by ROLLUP and CUBE

ROLLUP and **CUBE** are syntactic shortcuts that specify common grouping prefixes.

Group by ROLLUP

The **ROLLUP** operator requires an ordered list of grouping expressions to be supplied as arguments.

ROLLUP syntax.

```
SELECT ... [ GROUPING (column-name) ... ] ...
GROUP BY [ expression [, ...]
| ROLLUP ( expression [, ...] ) ]
```

GROUPING takes a column name as a parameter and returns a Boolean value as listed in the following table:

Table 1. Values returned by GROUPING with the ROLLUP operator

If the value of the result is	GROUPING returns
NULL created by a ROLLUP operation	1 (TRUE)
NULL indicating the row is a subtotal	1 (TRUE)
Not created by a ROLLUP operation	0 (FALSE)
A stored NULL	0 (FALSE)

ROLLUP first calculates the standard aggregate values specified in the **GROUP BY** clause. Then **ROLLUP** moves from right to left through the list of grouping columns and creates

progressively higher-level subtotals. A grand total is created at the end. If n is the number of grouping columns, then **ROLLUP** creates $n+1$ levels of subtotals.

This SQL Syntax...	Defines the Following Sets...
<code>GROUP BY ROLLUP (A, B, C);</code>	(A, B, C) (A, B) (A) ()

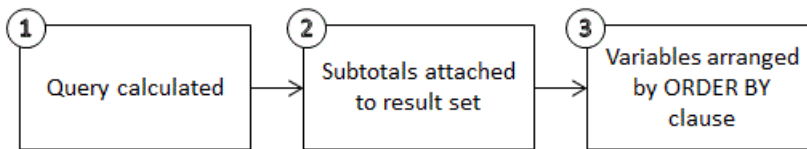
ROLLUP and subtotal rows

ROLLUP is equivalent to a **UNION** of a set of **GROUP BY** queries. The result sets of the following queries are identical. The result set of **GROUP BY** (A, B) consists of subtotals over all those rows in which A and B are held constant. To make a union possible, column C is assigned NULL.

This ROLLUP Query...	Is Equivalent to This Query Without ROLLUP...
<code>select year(orderdate) as year, quarter(orderdate) as Quarter, count(*) Orders from SalesOrders group by Rollup (year, quarter) order by year, quarter</code>	<code>Select null,null, count(*) Orders from SalesOrders union allSELECT year(orderdate) AS YEAR, NULL, count(*) Orders from SalesOrdersGROUP BY year(orderdate) union allSELECT year(orderdate) as YEAR, quarter(orderdate) as QUATER, count(*) Orders from SalesOrdersGROUP BY year(orderdate), quarter(orderdate)</code>

Subtotal rows can help you analyze data, especially if there are large amounts of data, different dimensions to the data, data contained in different tables, or even different databases altogether. For example, a sales manager might find reports on sales figures broken down by sales representative, region, and quarter to be useful in understanding patterns in sales. Subtotals for the data give the sales manager a picture of overall sales from different perspectives. Analyzing this data is easier when summary information is provided based on the criteria that the sales manager wants to compare.

With OLAP, the procedure for analyzing and computing row and column subtotals is invisible to users.

Figure 2: Subtotals

1. This step yields an intermediate result set that has not yet considered the **ROLLUP**.
2. Subtotals are evaluated and attached to the result set.
3. The rows are arranged according to the **ORDER BY** clause in the query.

NULL values and subtotal rows

When rows in the input to a **GROUP BY** operation contain NULL, there is the possibility of confusion between subtotal rows added by the **ROLLUP** or **CUBE** operations and rows that contain NULL values that are part of the original input data.

The Grouping() function distinguishes subtotal rows from others by taking a column in the **GROUP BY** list as its argument, and returning 1 if the column is NULL because the row is a subtotal row, and 0 otherwise.

The following example includes Grouping() columns in the result set. Rows are highlighted that contain NULL as a result of the input data, not because they are subtotal rows. The Grouping() columns are highlighted. The query is an outer join between the Employees table and the SalesOrders table. The query selects female employees who live in Texas, New York, or California. NULL appears in the columns corresponding to those female employees who are not sales representatives (and therefore have no sales).

Note: For examples, use the SAP Sybase IQ demo database iqdemo.db.

```

SELECT Employees.EmployeeID as EMP, year(OrderDate) as
YEAR, count(*) as ORDERS, grouping(EMP) as
GE, grouping(YEAR) as GY
FROM Employees LEFT OUTER JOIN SalesOrders on
Employees.EmployeeID = SalesOrders.SalesRepresentative
WHERE Employees.Sex IN ('F') AND Employees.State
IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (YEAR, EMP)
ORDER BY YEAR, EMP
  
```

The preceding query returns:

EMP	YEAR	ORDERS	GE	GY
NULL	NULL	5	1	0
NULL	NULL	169	1	1
102	NULL	1	0	0
309	NULL	1	0	0
1062	NULL	1	0	0
1090	NULL	1	0	0
1507	NULL	1	0	0
NULL	2000	98	1	0

667	2000	34	0	0
949	2000	31	0	0
1142	2000	33	0	0
NULL	2001	66	1	0
667	2001	20	0	0
949	2001	22	0	0
1142	2001	24	0	0

For each prefix, a *subtotal row* is constructed that corresponds to all rows in which the prefix columns have the same value.

To demonstrate **ROLLUP** results, examine the example query again:

```
SELECT year (OrderDate) AS Year, quarter
  (OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
  GROUP BY ROLLUP (Year, Quarter)
  ORDER BY Year, Quarter
```

In this query, the prefix containing the `Year` column leads to a summary row for `Year=2000` and a summary row for `Year=2001`. A single summary row for the prefix has no columns, which is a subtotal over all rows in the intermediate result set.

The value of each column in a subtotal row is as follows:

- Column included in the prefix – the value of the column. For example, in the preceding query, the value of the `Year` column for the subtotal over rows with `Year=2000` is 2000.
- Column excluded from the prefix – NULL. For example, the `Quarter` column has a value of NULL for the subtotal rows generated by the prefix consisting of the `Year` column.
- Aggregate function – an aggregate over the values of the excluded columns. Subtotal values are computed over the rows in the underlying data, not over the aggregated rows. In many cases, such as **SUM** or **COUNT**, the result is the same, but the distinction is important in the case of statistical functions such as **AVG**, **STDDEV**, and **VARIANCE**, for which the result differs.

Restrictions on the **ROLLUP** operator are:

- The **ROLLUP** operator supports all of the aggregate functions available to the **GROUP BY** clause except **COUNT DISTINCT** and **SUM DISTINCT**.
- **ROLLUP** can only be used in the **SELECT** statement; you cannot use **ROLLUP** in a subquery.
- A grouping specification that combines multiple **ROLLUP**, **CUBE**, and **GROUP BY** columns in the same **GROUP BY** clause is not currently supported.
- Constant expressions as **GROUP BY** keys are not supported.

ROLLUP example 2—The following example illustrates the use of **ROLLUP** and **GROUPING** and displays a set of mask columns created by **GROUPING**. The digits 0 and 1 displayed in columns `S`, `N`, and `C` are the values returned by **GROUPING** to represent the value of the **ROLLUP** result. A program can analyze the results of this query by using a mask of “011” to identify subtotal rows and “111” to identify the row of overall totals.

```
SELECT size, name, color, SUM(quantity),
       GROUPING(size) AS S,
       GROUPING(name) AS N,
       GROUPING(color) AS C
FROM Products
GROUP BY ROLLUP(size, name, color) HAVING (S=1 or N=1 or C=1)
ORDER BY size, name, color;
```

The preceding query returns:

size	name	color	SUM	S	N	C
(NULL)	(NULL)	(NULL)	496	1	1	1
Large	(NULL)	(NULL)	71	0	1	1
Large	Sweatshirt	(NULL)	71	0	0	1
Medium	(NULL)	(NULL)	134	0	1	1
Medium	Shorts	(NULL)	80	0	0	1
Medium	Tee Shirt	(NULL)	54	0	0	1
One size fits all	(NULL)	(NULL)	263	0	1	1
One size fits all	Baseball Cap	(NULL)	124	0	0	1
One size fits all	Tee Shirt	(NULL)	75	0	0	1
One size fits all	Visor	(NULL)	64	0	0	1
Small	(NULL)	(NULL)	28	0	1	1
Small	Tee Shirt	(NULL)	28	0	0	1

Note: In the Rollup Example 2 results, the SUM column displays as SUM(products.quantity).

ROLLUP example 3—The following example illustrates the use of **GROUPING** to distinguish stored NULL values and “NULL” values created by the **ROLLUP** operation. Stored NULL values are then displayed as [NULL] in column prod_id, and “NULL” values created by **ROLLUP** are replaced with ALL in column PROD_IDS, as specified in the query.

```
SELECT year(ShipDate) AS Year,
       ProductID, SUM(quantity) AS OSum,
CASE
    WHEN GROUPING(Year) = 1
    THEN 'ALL'
    ELSE
    CAST(Year AS char(8))
END,
CASE
    WHEN GROUPING(ProductID) = 1
    THEN 'ALL'
    ELSE
    CAST(ProductID as char(8))
END
FROM SalesOrderItems
GROUP BY ROLLUP(Year, ProductID) HAVING OSum > 36
ORDER BY Year, ProductID;
```

The preceding query returns:

Year	ProductID	OSum	...(Year)...	...(ProductID)...
-----	-----	-----	-----	-----

NULL	NULL	28359	ALL	ALL
2000	NULL	17642	2000	ALL
2000	300	1476	2000	300
2000	301	1440	2000	301
2000	302	1152	2000	302
2000	400	1946	2000	400
2000	401	1596	2000	401
2000	500	1704	2000	500
2000	501	1572	2000	501
2000	600	2124	2000	600
2000	601	1932	2000	601
2000	700	2700	2000	700
2001	NULL	10717	2001	ALL
2001	300	888	2001	300
2001	301	948	2001	301
2001	302	996	2001	302
2001	400	1332	2001	400
2001	401	1105	2001	401
2001	500	948	2001	500
2001	501	936	2001	501
2001	600	936	2001	600
2001	601	792	2001	601
2001	700	1836	2001	700

ROLLUP example 4—The next example query returns data that summarizes the number of sales orders by year and quarter.

```
SELECT year (OrderDate) AS Year,
quarter(OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
GROUP BY ROLLUP (Year, Quarter)
ORDER BY Year, Quarter
```

The following figure illustrates the query results with subtotal rows highlighted in the result set. Each subtotal row contains a NULL value in the column or columns over which the subtotal is computed.

	Year	Quarter	Orders
①	(NULL)	(NULL)	648
②	2000	(NULL)	380
③	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
②	2001	(NULL)	268
③	2001	1	139
	2001	2	119
	2001	3	10

Row [1] represents the total number of orders across both years (2000, 2001) and all quarters. This row contains NULL in both the `Year` and `Quarter` columns and is the row where all columns were excluded from the prefix.

Note: Every **ROLLUP** operation returns a result set with one row where NULL appears in each column except for the aggregate column. This row represents the summary of each column to the aggregate function. For example, if SUM were the aggregate function in question, this row would represent the grand total of all values.

Row [2] represent the total number of orders in the years 2000 and 2001, respectively. Both rows contain NULL in the `Quarter` column because the values in that column are rolled up to give a subtotal for `Year`. The number of rows like this in your result set depends on the number of variables that appear in your **ROLLUP** query.

The remaining rows marked [3] provide summary information by giving the total number of orders for each quarter in both years.

ROLLUP example 5—This example of the **ROLLUP** operation returns a slightly more complicated result set, which summarizes the number of sales orders by year, quarter, and region. In this example, only the first and second quarters and two selected regions (Canada and the Eastern region) are examined.

```
SELECT year(OrderDate) AS Year, quarter(OrderDate) AS Quarter,
region, COUNT(*) AS Orders
FROM SalesOrders WHERE region IN ('Canada','Eastern') AND quarter IN
(1, 2)
GROUP BY ROLLUP (Year, Quarter, Region) ORDER BY Year, Quarter, Region
```

The following figure illustrates the result set from the above query. Each subtotal row contains a NULL in the column or columns over which the subtotal is computed.

	Year	Quarter	Region	Orders
① →	(NULL)	(NULL)	(NULL)	183
	2000	(NULL)	(NULL)	68
	2000	1	(NULL)	36
	2000	1	Canada	3
②	2000	1	Eastern	33
	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	(NULL)	(NULL)	115
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

Row [1] is an aggregate over all rows and contains NULL in the Year, Quarter, and Region columns. The value in the Orders column of this row represents the total number of orders in Canada and the Eastern region in quarters 1 and 2 in the years 2000 and 2001.

The rows marked [2] represent the total number of sales orders in each year (2000) and (2001) in quarters 1 and 2 in Canada and the Eastern region. The values of these rows [2] are equal to the grand total represented in row [1].

The rows marked [3] provide data about the total number of orders for the given year and quarter by region.

	Year	Quarter	Region	Orders
	(NULL)	(NULL)	(NULL)	183
	2000	(NULL)	(NULL)	68
3	2000	1	(NULL)	36
	2000	1	Canada	3
	2000	1	Eastern	33
	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	(NULL)	(NULL)	115
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

The rows marked [4] provide data about the total number of orders for each year, each quarter, and each region in the result set.

	Year	Quarter	Region	Orders
	(NULL)	(NULL)	(NULL)	183
	2000	(NULL)	(NULL)	68
	2000	1	(NULL)	36
4	2000	1	Canada	3
	2000	1	Eastern	33
	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	(NULL)	(NULL)	115
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

Group by CUBE

The **CUBE** operator in the **GROUP BY** clause analyzes data by forming the data into groups in more than one dimension (grouping expression).

CUBE requires an ordered list of dimensions as arguments and enables the **SELECT** statement to calculate subtotals for all possible combinations of the group of dimensions that you specify in the query and generates a result set that shows aggregates for all combinations of values in selected columns.

CUBE syntax:

```
SELECT ... [ GROUPING (column-name) ... ] ...
GROUP BY [ expression [,...]
| CUBE ( expression [,...] ) ]
```

GROUPING takes a column name as a parameter, and returns a Boolean value as listed in the following table:

Table 2. Values returned by GROUPING with the CUBE operator

If the value of the result is	GROUPING returns
NULL created by a CUBE operation	1 (TRUE)
NULL indicating the row is a subtotal	1 (TRUE)
Not created by a CUBE operation	0 (FALSE)
A stored NULL	0 (FALSE)

CUBE is particularly useful when your dimensions are not a part of the same hierarchy.

This SQL syntax...	Defines the following sets...
GROUP BY CUBE (A, B, C);	(A, B, C) (A, B) (A, C) (A) (B, C) (B) (C) ()

Restrictions on the **CUBE** operator are:

- The **CUBE** operator supports all of the aggregate functions available to the **GROUP BY** clause, but **CUBE** is currently not supported with **COUNT DISTINCT** or **SUM DISTINCT**.

- **CUBE** is currently not supported with the inverse distribution analytical functions, **PERCENTILE_CONT** and **PERCENTILE_DISC**.
- **CUBE** can only be used in the **SELECT** statement; you cannot use **CUBE** in a **SELECT** subquery.
- A **GROUPING** specification that combines **ROLLUP**, **CUBE**, and **GROUP BY** columns in the same **GROUP BY** clause is not currently supported.
- Constant expressions as **GROUP BY** keys are not supported.

Note: **CUBE** performance diminishes if the size of the cube exceeds the size of the temp cache.

GROUPING can be used with the **CUBE** operator to distinguish between stored NULL values and NULL values in query results created by **CUBE**.

See the examples in the description of the **ROLLUP** operator for illustrations of the use of the **GROUPING** function to interpret results.

All **CUBE** operations return result sets with at least one row where NULL appears in each column except for the aggregate columns. This row represents the summary of each column to the aggregate function.

CUBE example 1—The following queries use data from a census, including the state (geographic location), gender, education level, and income of people. The first query contains a **GROUP BY** clause that organizes the results of the query into groups of rows, according to the values of the columns `state`, `gender`, and `education` in the table `census` and computes the average income and the total counts of each group. This query uses only the **GROUP BY** clause without the **CUBE** operator to group the rows.

```
SELECT State, Sex as gender, DepartmentID,
COUNT(*) ,CAST(ROUND(AVG(Salary),2) AS NUMERIC(18,2)) AS AVERAGEFROM
employees WHERE state IN ('MA' , 'CA')GROUP BY State, Sex,
DepartmentIDORDER BY 1,2;
```

The results from the above query:

state	gender	DepartmentID	COUNT()	AVERAGE
CA	F	200	2	58650.00
CA	M	200	1	39300.00

Use the **CUBE** extension of the **GROUP BY** clause, if you want to compute the average income in the entire census of state, gender, and education and compute the average income in all possible combinations of the columns `state`, `gender`, and `education`, while making only a single pass through the census data. For example, use the **CUBE** operator if you want to compute the average income of all females in all states, or compute the average income of all people in the census according to their education and geographic location.

When **CUBE** calculates a group, a NULL value is generated for the columns whose group is calculated. The **GROUPING** function must be used to distinguish whether a NULL is a NULL stored in the database or a NULL resulting from **CUBE**. The **GROUPING** function returns 1 if the designated column has been merged to a higher level group.

CUBE example 2—The following query illustrates the use of the **GROUPING** function with **GROUP BY CUBE**.

```
SELECT case grouping(State) WHEN 1 THEN 'ALL' ELSE StateEND AS
c_state, case grouping(sex) WHEN 1 THEN 'ALL'ELSE Sex end AS
c_gender, case grouping(DepartmentID)WHEN 1 THEN 'ALL' ELSE
cast(DepartmentID as char(4)) endAS c_dept, COUNT(*),
CAST(ROUND(AVG(salary),2) ASNUMERIC(18,2))AS AVERAGEFROM employees
WHERE state IN ('MA' , 'CA')GROUP BY CUBE(state, sex,
DepartmentID)ORDER BY 1,2,3;
```

The results of this query are shown below. The NULLs generated by **CUBE** to indicate a subtotal row are replaced with ALL in the subtotal rows, as specified in the query.

c_state	c_gender	c_dept	COUNT()	AVERAGE
ALL	ALL	200	3	52200.00
ALL	ALL	ALL	3	52200.00
ALL	F	200	2	58650.00
ALL	F	ALL	2	58650.00
ALL	M	200	1	39300.00
ALL	M	ALL	1	39300.00
CA	ALL	200	3	52200.00
CA	ALL	ALL	3	52200.00
CA	F	200	2	58650.00
CA	F	ALL	2	58650.00
CA	M	200	1	39300.00
CA	M	ALL	1	39300.00

CUBE example 3—In this example, the query returns a result set that summarizes the total number of orders and then calculates subtotals for the number of orders by year and quarter.

Note: As the number of variables that you want to compare increases, the cost of computing the cube increases exponentially.

```
SELECT year (OrderDate) AS Year, quarter(OrderDate) AS Quarter, COUNT
(*) OrdersFROM SalesOrdersGROUP BY CUBE (Year, Quarter)ORDER BY Year,
Quarter
```

The figure that follows represents the result set from the query. The subtotal rows are highlighted in the result set. Each subtotal row has a NULL in the column or columns over which the subtotal is computed.

	Year	Quarter	Orders
①	(NULL)	(NULL)	648
②	(NULL)	1	226
	(NULL)	2	196
	(NULL)	3	101
	(NULL)	4	125
③	2000	(NULL)	380
	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
③	2001	(NULL)	268
	2001	1	139
	2001	2	119
	2001	3	10

The first highlighted row [1] represents the total number of orders across both years and all quarters. The value in the `Orders` column is the sum of the values in each of the rows marked [3]. It is also the sum of the four values in the rows marked [2].

The next set of highlighted rows [2] represents the total number of orders by quarter across both years. The two rows marked by [3] represent the total number of orders across all quarters for the years 2000 and 2001, respectively.

Analytical Functions

SAP Sybase IQ offers both simple and windowed aggregation functions that offer the ability to perform complex data analysis within a single SQL statement.

You can use these functions to compute results for queries such as “What is the quarterly moving average of the Dow Jones Industrial average,” or “List all employees and their cumulative salaries for each department.” Moving averages and cumulative sums can be calculated over various intervals, and aggregations and ranks can be partitioned, so aggregate calculation is reset when partition values change. Within the scope of a single query expression, you can define several different OLAP functions, each with its own arbitrary partitioning rules. Analytical functions can be broken into two categories:

- Simple aggregate functions, such as **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM** summarize data over a group of rows from the database. The groups are formed using the **GROUP BY** clause of the **SELECT** statement.
- Unary statistical aggregate functions that take one argument include **STDDEV**, **STDDEV_SAMP**, **STDDEV_POP**, **VARIANCE**, **VAR_SAMP**, and **VAR_POP**.

Both the simple and unary categories of aggregates summarize data over a group of rows from the database and can be used with a window specification to compute a moving window over a result set as it is processed.

Note: The aggregate functions **AVG**, **SUM**, **STDDEV**, **STDDEV_POP**, **STDDEV_SAMP**, **VAR_POP**, **VAR_SAMP**, and **VARIANCE** do not support binary data types **BINARY** and **VARBINARY**.

Simple Aggregate Functions

Simple aggregate functions, such as **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM** summarize data over a group of rows from the database.

The groups are formed using the **GROUP BY** clause of the **SELECT** statement. These aggregates are allowed only in the select list and in the **HAVING** and **ORDER BY** clauses of a **SELECT** statement.

Note: With the exception of Grouping() functions, both the simple and unary aggregates can be used in a windowing function that incorporates a <window clause> in a SQL query specification (a *window*) that conceptually creates a moving window over a result set as it is processed.

Windowing

A major feature of the ANSI SQL extensions for OLAP is a construct called a *window*. This windowing extension lets users divide result sets of a query (or a logical partition of a query) into groups of rows called partitions and determine subsets of rows to aggregate with respect to the current row.

You can use three classes of window functions with a window: ranking functions, the row numbering function, and window aggregate functions.

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

Windowing extensions specify a window function type over a window name or specification and are applied to partitioned result sets within the scope of a single query expression. A window partition is a subset of rows returned by a query, as defined by one or more columns in a special **OVER** clause:

```
olap_function() OVER (PARTITION BY col1, col2...)
```

Windowing operations let you establish information such as the ranking of each row within its partition, the distribution of values in rows within a partition, and similar operations.

Windowing also lets you compute moving averages and sums on your data, enhancing the ability to evaluate your data and its impact on your operations.

An OLAP window's three essential parts

The OLAP windows comprise three essential aspects: window partitioning, window ordering, and window framing. Each has a significant impact on the specific rows of data visible in a window at any point in time. Meanwhile, the OLAP **OVER** clause differentiates OLAP functions from other analytic or reporting functions with three distinct capabilities:

- Defining window partitions (**PARTITION BY** clause).
- Ordering rows within partitions (**ORDER BY** clause).
- Defining window frames (ROWS/RANGE specification).

To specify multiple windows functions, and to avoid redundant window definitions, you can specify a name for an OLAP window specifications. In this usage, the keyword, **WINDOW**, is followed by at least one window definition, separated by commas. A window definition includes the name by which the window is known in the query and the details from the windows specification, which lets you to define window partitioning, ordering, and framing:

```
<WINDOW CLAUSE> ::= <WINDOW DEFINITION LIST>
```

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
    } . . . ]
```

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

For each row in a window partition, users can define a window frame, which may vary the specific range of rows used to perform any computation on the current row of the partition. The current row provides the reference point for determining the start and end points of the window frame.

Window specifications can be based on either a physical number of rows using a window specification that defines a window frame unit of ROWS or a logical interval of a numeric value, using a window specification that defines a window frame unit of RANGE.

Within OLAP windowing operations, you can use the following functional categories:

- Ranking functions
- Windowing aggregate functions
- Statistical aggregate functions
- Distribution functions

Window Partitioning

Window partitioning is the division of user-specified result sets (input rows) using a **PARTITION BY** clause.

A partition is defined by one or more value expressions separated by commas. Partitioned data is also implicitly sorted and the default sort order is ascending (ASC).

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

If a window partition clause is not specified, then the input is treated as single partition.

Note: The term *partition* as used with analytic functions, refers only to dividing the set of result rows using a **PARTITION BY** clause.

A window partition can be defined based on an arbitrary expression. Also, because window partitioning occurs after **GROUPING** (if a **GROUP BY** clause is specified), the result of any aggregate function, such as **SUM**, **AVG**, and **VARIANCE**, can be used in a partitioning expression. Therefore, partitions provide another opportunity to perform grouping and ordering operations in addition to the **GROUP BY** and **ORDER BY** clauses; for example, you can construct queries that compute aggregate functions over aggregate functions, such as the maximum **SUM** of a particular quantity.

You can specify a **PARTITION BY** clause, even if there is no **GROUP BY** clause.

Window Ordering

Window ordering is the arrangement of results (rows) within each window partition using a window order clause, which contains one or more value expressions separated by commas.

If a window order clause is not specified, the input rows could be processed in an arbitrary order.

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

The OLAP window order clause is different from the **ORDER BY** clause that can be appended to a nonwindowed query expression.

The **ORDER BY** clause in an OLAP function, for example, typically defines the expressions for sorting rows within window partitions; however, you can use the **ORDER BY** clause without a **PARTITION BY** clause, in which case the sort specification ensures that the OLAP function is applied to a meaningful (and intended) ordering of the intermediate result set.

An order specification is a prerequisite for the ranking family of OLAP functions; it is the **ORDER BY** clause, not an argument to the function itself, that identifies the measures for the ranking values. In the case of OLAP aggregates, the **ORDER BY** clause is not required in general, but it is a prerequisite to defining a window frame. This is because the partitioned rows must be sorted before the appropriate aggregate values can be computed for each frame.

The **ORDER BY** clause includes semantics for defining ascending and descending sorts, as well as rules for the treatment of NULL values. By default, OLAP functions assume an ascending order, where the lowest measured value is ranked 1.

Although this behavior is consistent with the default behavior of the **ORDER BY** clause that ends a **SELECT** statement, it is counterintuitive for most sequential calculations. OLAP calculations often require a descending order, where the highest measured value is ranked 1; this requirement must be explicitly stated in the **ORDER BY** clause with the DESC keyword.

Note: Ranking functions require a <window order clause> because they are defined only over sorted input. As with an <order by clause> in a <query specification>, the default sort sequence is ascending.

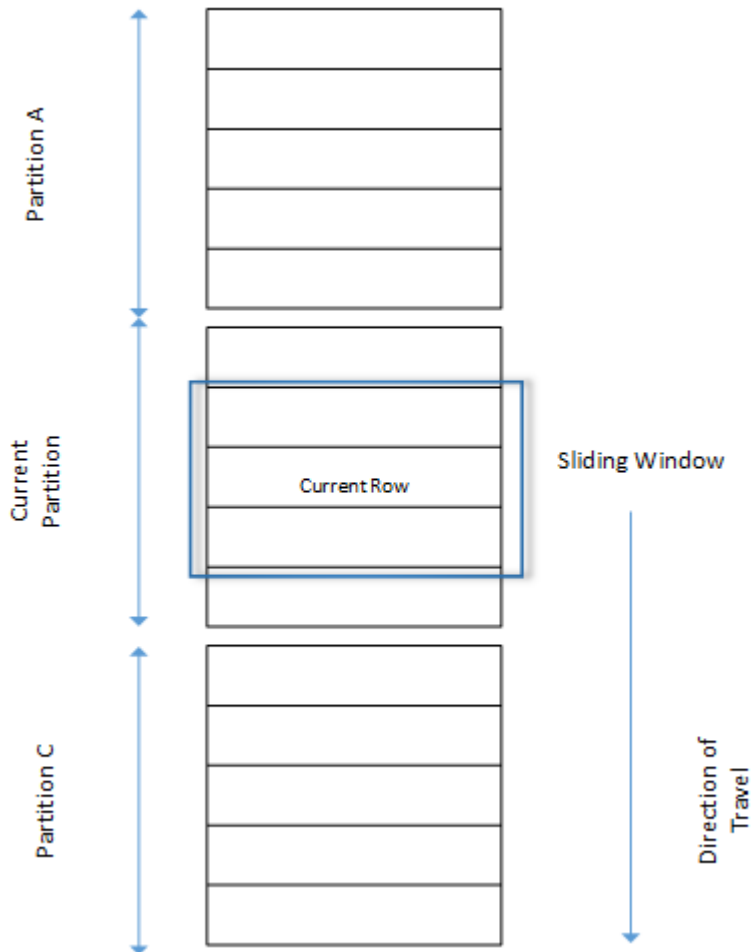
The use of a <window frame unit> of RANGE also requires the existence of a <window order clause>. In the case of RANGE, the <window order clause> may only consist of a single expression.

Window Framing

For nonranking aggregate OLAP functions, you can define a window frame with a window frame clause, which specifies the beginning and end of the window relative to the current row.

```
<WINDOW FRAME CLAUSE> ::=  
  <WINDOW FRAME UNIT>  
  <WINDOW FRAME EXTENT>
```

This OLAP function is computed with respect to the contents of a moving frame rather than the fixed contents of the whole partition. Depending on its definition, the partition has a start row and an end row, and the window frame slides from the starting point to the end of the partition.

Figure 3: Three-row moving window with partitioned input***UNBOUNDED PRECEDING and FOLLOWING***

Window frames can be defined by an unbounded aggregation group that either extends back to the beginning of the partition (**UNBOUNDED PRECEDING**) or extends to the end of the partition (**UNBOUNDED FOLLOWING**), or both.

UNBOUNDED PRECEDING includes all rows within the partition preceding the current row, which can be specified with either **ROWS** or **RANGE**. **UNBOUNDED FOLLOWING** includes all rows within the partition following the current row, which can be specified with either **ROWS** or **RANGE**.

The value **FOLLOWING** specifies either the range or number of rows following the current row. If **ROWS** is specified, then the value is a positive integer indicating a number of rows. If **RANGE** is specified, the window includes any rows that are less than the current row plus the

specified numeric value. For the RANGE case, the data type of the windowed value must be comparable to the type of the sort key expression of the **ORDER BY** clause. There can be only one sort key expression, and the data type of the sort key expression must allow addition.

The value PRECEDING specifies either the range or number of rows preceding the current row. If ROWS is specified, then the value is a positive integer indicating a number of rows. If RANGE is specified, the window includes any rows that are less than the current row minus the specified numeric value. For the RANGE case, the data type of the windowed value must be comparable to the type of the sort key expression of the **ORDER BY** clause. There can be only one sort key expression, and the data type of the sort key expression must allow subtraction. This clause cannot be specified in second bound group if the first bound group is CURRENT ROW or value FOLLOWING.

The combination BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING provides an aggregate over an entire partition, without the need to construct a join to a grouped query. An aggregate over an entire partition is also known as a reporting aggregate.

CURRENT ROW concept

In physical aggregation groups, rows are included or excluded based on their position relative to the current row, by counting adjacent rows. The current row is simply a reference to the next row in a query's intermediate results. As the current row advances, the window is reevaluated based on the new set of rows that lie within the window. There is no requirement that the current row be included in a window.

If a window frame clause is not specified, the default window frame depends on whether or not a window order clause is specified:

- If the window specification contains a window order clause, the window's start point is **UNBOUNDED PRECEDING**, and the end point is **CURRENT ROW**, thus defining a varying-size window suitable for computing cumulative values.
- If the window specification does not contain a window order clause, the window's start point is **UNBOUNDED PRECEDING**, and the end point is **UNBOUNDED FOLLOWING**, thus defining a window of fixed size, regardless of the current row.

Note: A window frame clause cannot be used with a ranking function.

You can also define a window by specifying a window frame unit that is row-based (rows specification) or value-based (range specification).

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

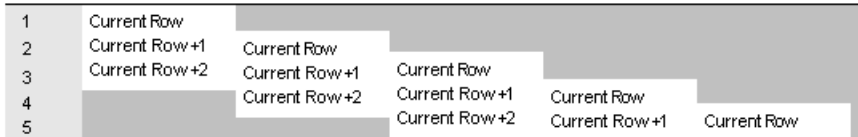
```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME BETWEEN>
```

When a window frame extent specifies **BETWEEN**, it explicitly provides the beginning and end of a window frame.

If the window frame extent specifies only one of these two values then the other value defaults to **CURRENT ROW**.

Row-based window frames—In the example rows [1] through [5] represent a partition; each row becomes the current row as the OLAP window frame slides forward. The frame is defined as *Between Current Row And 2 Following*, so each frame includes a maximum of three rows and a minimum of one row. When the frame reaches the end of the partition, only the current row is included. The shaded areas indicate which rows are excluded from the frame at each step.

Figure 4: Row-based window frames



The window frame imposes the following rules:

- When row [1] is the current row, rows [4] and [5] are excluded.
- When row [2] is the current row, rows [5] and [1] are excluded.
- When row [3] is the current row, rows [1] and [2] are excluded.
- When row [4] is the current row, rows [1], [2], and [3] are excluded.
- When row [5] is the current row, rows [1], [2], [3], and [4] are excluded.

The following diagram applies these rules to a specific set of values, showing the OLAP **AVG** function that would be calculated for each row. The sliding calculations produce a moving average with an interval of three rows or fewer, depending on which row is the current row:

Row	Dimension	Measure	OLAP_A V G
1	A	10	53.3
2	A	50	
3	A	100	
4	A	120	240
5	A	500	310
			500

The following example demonstrates a sliding window:

```
SELECT dimension, measure,
       AVG(measure) OVER(partition BY dimension
                        ORDER BY measure
                        ROWS BETWEEN CURRENT ROW and 2 FOLLOWING)
       AS olap_avg
FROM ...
```

The averages are computed as follows:

- Row [1] = (10 + 50 + 100)/3

- Row [2] = (50+ 100 + 120)/3
- Row [3] = (100 + 120 + 500)/3
- Row [4] = (120 + 500 + NULL)/3
- Row [5] = (500 + NULL + NULL)/3

Similar calculations would be computed for all subsequent partitions in the result set (such as, B, C, and so on).

If there are no rows in the current window, the result is NULL, except for **COUNT**.

ROWS

The window frame unit **ROWS** defines a window in the specified number of rows before or after the current row, which serves as the reference point that determines the start and end of a window.

Each analytical calculation is based on the current row within a partition. To produce determinative results for a window expressed in rows, the ordering expression should be unique.

The reference point for all window frames is the current row. The SQL/OLAP syntax provides mechanisms for defining a row-based window frame as any number of rows preceding or following the current row or preceding and following the current row.

The following list illustrates common examples of a window frame unit:

- Rows between unbounded preceding and current row – specifies a window whose start point is the beginning of each partition and the end point is the current row and is often used to construct windows that compute cumulative results, such as cumulative sums.
- Rows between unbounded preceding and unbounded following – specifies a fixed window, regardless of the current row, over the entire partition. The value of a window aggregate function is, therefore, identical in each row of the partition.
- Rows between 1 preceding and 1 following – specifies a fixed-sized moving window over three adjacent rows, one each before and after the current row. You can use this window frame unit to compute, for example, a 3-day or 3-month moving average.

Be aware of meaningless results that may be generated by gaps in the windowed values when using **ROWS**. If the set of values is not continuous, consider using **RANGE** instead of **ROWS**, because a window definition based on **RANGE** automatically handles adjacent rows with duplicate values and does not include other rows when there are gaps in the range.

Note: In the case of a moving window, it is assumed that rows containing NULL values exist before the first row, and after the last row, in the input. This means that in a 3-row moving window, the computation for the last row in the input—the current row— includes the immediately preceding row and a NULL value.

- Rows between current row and current row – restricts the window to the current row only.
- Rows between 1 preceding and 1 preceding – specifies a single row window consisting only of the preceding row, with respect to the current row. In combination with another

window function that computes a value based on the current row only, this construction makes it possible to easily compute deltas, or differences in value, between adjacent rows.

RANGE

Range-based window frames—The SQL/OLAP syntax supports another kind of window frame whose limits are defined in terms of a value-based—or range-based—set of rows, rather than a specific sequence of rows.

Value-based window frames define rows within a window partition that contain a specific range of numeric values. The OLAP function’s **ORDER BY** clause defines the numeric column to which the range specification is applied, relative to the current row’s value for that column. The range specification uses the same syntax as the rows specification, but the syntax is interpreted in a different way.

The window frame unit, **RANGE**, defines a window frame whose contents are determined by finding rows in which the ordering column has values within the specified range of value relative to the current row. This is called a logical offset of a window frame, which you can specify with constants, such as “3 preceding,” or any expression that can be evaluated to a numeric constant. When using a window defined with **RANGE**, there can be only a single numeric expression in the **ORDER BY** clause.

Note: **ORDER BY** key must be a numeric data in **RANGE** window frame

For example, a frame can be defined as the set of rows with *year* values some number of years preceding or following the current row’s year:

```
ORDER BY year ASC range BETWEEN 1 PRECEDING AND CURRENT ROW
```

The phrase 1 **PRECEDING** means the current row’s *year* value minus 1.

This kind of range specification is inclusive. If the current row’s *year* value is 2000, all rows in the window partition with year values 2000 and 1999 qualify for the frame, regardless of the physical position of those rows in the partition. The rules for including and excluding value-based rows are quite different from the rules applied to row-based frames, which depend entirely on the physical sequence of rows.

Put in the context of an OLAP **AVG()** calculation, the following partial result set further demonstrates the concept of a value-based window frame. Again, the frame consists of rows that:

- Have the same year as the current row
- Have the same year as the current row minus 1

Row	Dimension	Year	Measure	Olap_avg
1	A	1999	10000	10000
2	A	2001	5000	3000
3	A	2001	1000	3000
4	A	2002	12000	5250
5	A	2002	3000	5250

The following query demonstrates a range-based window definition:

Appendix: Using OLAP

```
SELECT dimension, year, measure,
       AVG(measure) OVER(PARTITION BY dimension
                        ORDER BY year ASC
                        range BETWEEN CURRENT ROW and 1 PRECEDING)
       as olap_avg
FROM ...
```

The averages are computed as follows:

- Row [1] = 1999; rows [2] through [5] are excluded; AVG = 10,000/1
- Row [2] = 2001; rows [1], [4], and [5] are excluded; AVG = 6,000/2
- Row [3] = 2001; rows [1], [4], and [5] are excluded; AVG = 6,000/2
- Row [4] = 2002; row [1] is excluded; AVG = 21,000/4
- Row [5] = 2002; row [1] is excluded; AVG = 21,000/4

Ascending and descending order for value -based frames—The **ORDER BY** clause for an OLAP function with a value-based window frame not only identifies the numeric column on which the range specification is based; it also declares the sort order for the **ORDER BY** values. The following specification is subject to the sort order that precedes it (ASC or DESC):

```
RANGE BETWEEN CURRENT ROW AND n FOLLOWING
```

The specification *n* FOLLOWING means:

- Plus *n* if the partition is sorted in default ascending order (ASC)
- Minus *n* if the partition is sorted in descending order (DESC)

For example, assume that the `year` column contains four distinct values, from 1999 to 2002. The following table shows the default ascending order of these values on the left and the descending order on the right:

ORDER BY year ASC	ORDER BY year DESC
1999	2002
2000	2001
2001	2000
2002	1999

If the current row is 1999 and the frame is specified as follows, rows that contain the values 1999 and 1998 (which does not exist in the table) are included in the frame:

```
ORDER BY year DESC range BETWEEN CURRENT ROW and 1 FOLLOWING
```

Note: The sort order of the **ORDER BY** values is a critical part of the test for qualifying rows in a value-based frame; the numeric values alone do not determine exclusion or inclusion.

Using an unbounded window—The following query produces a result set consisting of all of the products accompanied by the total quantity of all products:

```
SELECT id, description, quantity,
       SUM(quantity) OVER () AS total
FROM products;
```

Computing deltas between adjacent rows—Using two windows—one over the current row and the other over the previous row—provides a direct way of computing deltas, or changes, between adjacent rows.

```
SELECT EmployeeID, Surname, SUM(salary)
OVER(ORDER BY BirthDate rows between current row and current row)
AS curr, SUM(Salary)
OVER(ORDER BY BirthDate rows between 1 preceding and 1 preceding)
AS prev, (curr-prev) as delta
FROM Employees
WHERE State IN ('MA', 'AZ', 'CA', 'CO') AND DepartmentID>10
ORDER BY EmployeeID, Surname;
```

The results from the query:

EmployeeID	Surname	curr	prev	delta
148	Jordan	51432.000191		
209	Bertrand		29800.000	
39300.000		-9500.000278		
225	Melkisetian	48500.000	42300.000	
6200.000299				
657	Overbey		39300.000	
41700.750		-2400.750318		
902	Crow		41700.750	
45000.000		-3299.250586		
949	Coleman		42300.000	
46200.000		-3900.000690		
1053	Poitras		46200.000	
29800.000		16400.000703		
1090	Martinez		55500.800	51432.000
4068.800949				
1154	Savarino		72300.000	
55500.800		16799.2001101		
1420	Preston	37803.000		48500.000
-10697.0001142				
1507	Clark	45000.000		72300.000
-27300.000				

Although the window function **SUM()** is used, the sum contains only the salary value of either the current or previous row because of the way the window is specified. Also, the `prev` value of the first row in the result is `NULL` because it has no predecessor; therefore, the `delta` is `NULL` as well.

In each of the examples above, the function used with the **OVER()** clause is the **SUM()** aggregate function.

Explicit and Inline Window Clauses

SQL OLAP provides two ways of specifying a window in a query:

- The explicit window clause lets you define a window that follows a **HAVING** clause. You reference windows defined with those window clauses by specifying their names when you invoke an OLAP function, such as:

```
SUM ( ...) OVER w2
```

- The inline window specification lets you define a window in the **SELECT** list of a query expression. This capability lets you define your windows in a window clause that follows the **HAVING** clause and then reference them by name from your window function invocations, or to define them along with the function invocations.

Note: If you use an inline window specification, you cannot name the window. Two or more window function invocations in a single **SELECT** list that use identical windows must either reference a named window defined in a window clause or they must define their inline windows redundantly.

Window function example—The following example shows a window function. The query returns a result set that partitions the data by department and then provides a cumulative summary of employees’ salaries, starting with the employee who has been at the company the longest. The result set includes only those employees who reside in Massachusetts. The column `sum_salary` provides the cumulative total of employees’ salaries.

```
SELECT DepartmentID, Surname, StartDate, Salary, SUM(Salary) OVER
(PARTITION BY DepartmentID ORDER BY startdate
rows between unbounded preceding and current row)
AS sum_salary FROM Employees
WHERE State IN ('CA') AND DepartmentID IN (100, 200)
ORDER BY DepartmentID;
```

The following result set is partitioned by department.

DepartmentID	Surname	start_date	salary	sum_salary
200	Overbey	1987-02-19	39300.000	39300.000
200	Savarino	1989-11-07	72300.000	111600.000
200	Clark	1990-07-21	45000.000	156600.000

Ranking Functions

Ranking functions let you compile a list of values from the data set in ranked order, as well as compose single-statement SQL queries that fulfil requests such as, “Name the top 10 products shipped this year by total sales,” or “Give the top 5% of salespersons who sold orders to at least 15 different companies.”

SQL/OLAP defines five functions that are categorized as ranking functions:

```
<RANK FUNCTION TYPE> ::=
RANK | DENSE_RANK | PERCENT_RANK | ROW_NUMBER | NTILE
```

Ranking functions let you compute a rank value for each row in a result set based on the order specified in the query. For example, a sales manager might need to identify the top or bottom sales people in the company, the highest- or lowest-performing sales region, or the best- or worst-selling products. Ranking functions can provide this information.

RANK

The **RANK** function returns a number that indicates the rank of the current row among the rows in the row's partition, as defined by the **ORDER BY** clause.

The first row in a partition has a rank of 1, and the last rank in a partition containing 25 rows is 25. **RANK** is specified as a syntax transformation, which means that an implementation can choose to actually transform **RANK** into its equivalent, or it can merely return a result equivalent to the result that transformation would return.

In the following example, *ws1* indicates the window specification that defines the window named *w1*.

```
RANK () OVER ws
```

is equivalent to:

```
( COUNT (*) OVER ( ws RANGE UNBOUNDED PRECEDING )  
- COUNT (*) OVER ( ws RANGE CURRENT ROW ) + 1 )
```

The transformation of the **RANK** function uses logical aggregation (RANGE). As a result, two or more records that are tied—or have equal values in the ordering column—have the same rank. The next group in the partition that has a different value has a rank that is more than one greater than the rank of the tied rows. For example, if there are rows whose ordering column values are 10, 20, 20, 20, 30, the rank of the first row is 1 and the rank of the second row is 2. The rank of the third and fourth row is also 2, but the rank of the fifth row is 5. There are no rows whose rank is 3 or 4. This algorithm is sometimes known as sparse ranking.

RANK Function [Analytical]

Ranks items in a group.

Syntax

```
RANK () OVER ( [ PARTITION BY ] ORDER BY expression [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

INTEGER

Remarks

RANK is a rank analytical function. The rank of row *R* is defined as the number of rows that precede *R* and are not peers of *R*. If two or more rows are not distinct within the groups

specified in the **OVER** clause or distinct over the entire result set, then there are one or more gaps in the sequential rank numbering. The difference between **RANK** and **DENSE_RANK** is that **DENSE_RANK** leaves no gap in the ranking sequence when there is a tie. **RANK** leaves a gap when there is a tie.

RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY** for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

The **PARTITION BY** window partitioning clause in the **OVER (ORDER BY)** clause is optional.

The **ASC** or **DESC** parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **RANK** can be in a view or a union. The **RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server or SQL Anywhere.

Example

This statement illustrates the use of the **RANK** function:

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

The results from the above query:

Surname	Sex	Salary	RANK
-----	---	-----	----
Savarino	F	72300.000	1
Jordan	F	51432.000	2
Clark	F	45000.000	3
Coleman	M	42300.000	1
Overbey	M	39300.000	2

DENSE_RANK

DENSE_RANK returns ranking values without gaps.

The values for rows with ties are still equal, but the ranking of the rows represents the positions of the clusters of rows having equal values in the ordering column, rather than the positions of the individual rows. As in the **RANK** example, where rows ordering column values are 10, 20, 20, 20, 30, the rank of the first row is still 1 and the rank of the second row is still 2, as are the ranks of the third and fourth rows. The last row, however, is 3, not 5.

DENSE_RANK is computed through a syntax transformation, as well.

```
DENSE_RANK ( ) OVER ws
```

is equivalent to:

```
COUNT ( DISTINCT ROW ( expr_1, . . . , expr_n ) )  
OVER ( ws RANGE UNBOUNDED PRECEDING )
```

In the above example, *expr_1* through *expr_n* represent the list of value expressions in the sort specification list of window w1.

DENSE_RANK Function [Analytical]

Ranks items in a group.

Syntax

```
DENSE_RANK ( ) OVER ( ORDER BY expression [ ASC | DESC ] )
```

Parameters

Table 3. Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

INTEGER

Remarks

DENSE_RANK is a rank analytical function. The dense rank of row R is defined as the number of rows preceding and including R that are distinct within the groups specified in the **OVER** clause or distinct over the entire result set. The difference between **DENSE_RANK** and **RANK** is that **DENSE_RANK** leaves no gap in the ranking sequence when there is a tie. **RANK** leaves a gap when there is a tie.

DENSE_RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each

group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY** for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

The **ASC** or **DESC** parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

DENSE_RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **DENSE_RANK** can be in a view or a union. The **DENSE_RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server or SQL Anywhere.

Example

The following statement illustrates the use of the **DENSE_RANK** function:

```
SELECT s_suppkey, DENSE_RANK()
OVER ( ORDER BY ( SUM(s_acctBal) DESC )
AS rank_dense FROM supplier GROUP BY s_suppkey;
```

s_suppkey	sum_acctBal	rank_dense
supplier#011	200,000	1
supplier#002	200,000	1
supplier#013	123,000	2
supplier#004	110,000	3
supplier#035	110,000	3
supplier#006	50,000	4
supplier#021	10,000	5

PERCENT_RANK

The **PERCENT_RANK** function calculates a percentage for the rank, rather than a fractional amount, and returns a decimal value between 0 and 1.

PERCENT_RANK returns the relative rank of a row, which is a number that indicates the relative position of the current row within the window partition in which it appears. For example, in a partition that contains 10 rows having different values in the ordering columns, the third row is given a **PERCENT_RANK** value of 0.222 ..., because you have covered 2/9 (22.222...%) of rows following the first row of the partition. **PERCENT_RANK** of a row is defined as one less than the **RANK** of the row divided by one less than the number of rows in the partition, as seen in the following example (where “ANT” stands for an approximate numeric type, such as **REAL** or **DOUBLE PRECISION**).

```
PERCENT_RANK () OVER ws
```

is equivalent to:

```
CASE
  WHEN COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
    PRECEDING AND UNBOUNDED FOLLOWING ) = 1
  THEN CAST ( 0 AS ANT)
  ELSE
    ( CAST ( RANK () OVER ( ws ) AS ANT ) - 1 /
      ( COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
        PRECEDING AND UNBOUNDED FOLLOWING ) - 1 )
    )
END
```

PERCENT_RANK Function [Analytical]

Computes the (fractional) position of one row returned from a query with respect to the other rows returned by the query, as defined by the **ORDER BY** clause.

Returns a decimal value between 0 and 1.

Syntax

```
PERCENT_RANK () OVER ( ORDER BY expression [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

The **PERCENT_RANK** function returns a DOUBLE value between 0 and 1.

Remarks

PERCENT_RANK is a rank analytical function. The percent rank of a row R is defined as the rank of a row in the groups specified in the **OVER** clause minus one divided by the number of total rows in the groups specified in the **OVER** clause minus one. **PERCENT_RANK** returns a value between 0 and 1. The first row has a percent rank of zero.

The **PERCENT_RANK** of a row is calculated as

```
(Rx - 1) / (NtotalRow - 1)
```

where *R_x* is the rank position of a row in the group and *N_{totalRow}* is the total number of rows in the group specified by the **OVER** clause.

PERCENT_RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY**

for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

The **ASC** or **DESC** parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

PERCENT_RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **PERCENT_RANK** can be in a view or a union. The **PERCENT_RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server or SQL Anywhere.

Example

The following statement illustrates the use of the **PERCENT_RANK** function:

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;
```

s_suppkey	sum_acctBal	percent_rank_all
supplier#011	200000	0
supplier#002	200000	0
supplier#013	123000	0.3333
supplier#004	110000	0.5
supplier#035	110000	0.5
supplier#006	50000	0.8333
supplier#021	10000	1

ROW_NUMBER

The **ROW_NUMBER** function returns a unique row number for each row.

If you define window partitions, **ROW_NUMBER** starts the row numbering in each partition at 1, and increments each row by 1. If you do not specify a window partition, **ROW_NUMBER** numbers the complete result set from 1 to the total cardinality of the table.

The **ROW_NUMBER** function syntax is:

```
ROW_NUMBER () OVER ([PARTITION BY window partition] ORDER BY window ordering)
```

ROW_NUMBER does not require an argument, but you must specify the parentheses.

The **PARTITION BY** clause is optional. The **OVER (ORDER BY)** clause cannot contain a window frame **ROWS/RANGE** specification.

ROW_NUMBER Function [Analytical]

A ranking function that returns a unique row number for each row in a window partition, restarting the row numbering at the start of every window partition.

If no window partitions exist, the function numbers the rows in the result set from 1 to the cardinality of the table.

Syntax

```
ROW_NUMBER () OVER ([PARTITION BY window partition] ORDER BY window ordering)
```

Parameters

Parameter	Description
window partition	(Optional) One or more value expressions separated by commas indicating how you want to divide the set of result rows.
window ordering	Defines the expressions for sorting rows within window partitions, if specified, or within the result set if you did not specify a window partition.

Remarks

The **ROW_NUMBER** function requires an **OVER (ORDER BY)** window specification. The window partitioning clause in the **OVER (ORDER BY)** clause is optional. The **OVER (ORDER BY)** clause must not contain a window frame **ROWS/RANGE** specification.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T611.

Example

The following example returns salary data from the Employees table, partitions the result set by department ID, and orders the data according to employee start date. The **ROW_NUMBER** function assigns each row a row number, and restarts the row numbering for each window partition:

```
SELECT DepartmentID dID, StartDate, Salary,
ROW_NUMBER() OVER (PARTITION BY dID ORDER BY StartDate) FROM Employees
ORDER BY 1,2;
```

The returned result set is:

```
dID      StartDate      Salary      Row_number()
=====
=====
```

100	1986-10-14	42,998.000	1
100	1987-07-23	39,875.500	2
100	1988-03-23	37,400.000	3
100	1989-04-20	42,500.000	4
100	1990-01-15	42,100.000	5
200	1985-02-03	38,500.000	1
200	1987-02-19	39,300.000	2
200	1988-11-22	39,800.000	3
200	1989-06-01	34,892.000	4
200	1990-05-13	33,890.000	5
200	1990-07-11	37,803.000	6

Ranking Examples

These are some of the ranking functions examples:

Ranking example 1—The SQL query that follows finds the male and female employees from California, and ranks them in descending order according to salary.

```
SELECT Surname, Sex, Salary, RANK() OVER (
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA') AND DepartmentID =200
ORDER BY Salary DESC;
```

The results from the above query:

Surname	Sex	Salary	RANK
-----	---	-----	----
Savarino	F	72300.000	1
Clark	F	45000.000	2
Overbey	M	39300.000	3

Ranking example 2—Using the query from the previous example, you can change the data by partitioning it by gender. The following example ranks employees in descending order by salary and partitions by gender:

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

The results from the above query:

Surname	Sex	Salary	RANK
-----	---	-----	----
Savarino	F	72300.000	1
Jordan	F	51432.000	2
Clark	F	45000.000	3
Coleman	M	42300.000	1
Overbey	M	39300.000	2

Ranking example 3—This example ranks a list of female employees in California and Texas in descending order according to salary. The **PERCENT_RANK** function provides the cumulative total in descending order.


```
SELECT Surname, Salary, Sex, CAST(PERCENT_RANK() OVER
(ORDER BY Salary DESC) AS numeric (4, 2)) AS RANK
FROM Employees WHERE State IN ('CA', 'TX') AND Sex ='F'
ORDER BY Salary DESC;
```

The results from the above query:

Surname	salary	sex	RANK
Savarino	72300.000	F	0.00
Smith	51411.000	F	0.33
Clark	45000.000	F	0.66
Garcia	39800.000	F	1.00

Ranking example 4—You can use the **PERCENT_RANK** function to find the top or bottom percentiles in the data set. This query returns male employees whose salary is in the top five percent of the data set.

```
SELECT * FROM (SELECT Surname, Salary, Sex,
CAST(PERCENT_RANK() OVER (ORDER BY salary DESC) as
numeric (4, 2)) AS percent
FROM Employees WHERE State IN ('CA') AND sex ='F' ) AS
DT where percent > 0.5
ORDER BY Salary DESC;
```

The results from the above query:

Surname	salary	sex	percent
Clark	45000.000	F	1.00

Ranking example 5—This example uses the **ROW_NUMBER** function to return row numbers for each row in all window partitions. The query partitions the `Employees` table by department ID, and orders the rows in each partition by start date.

```
SELECT DepartmentID dID, StartDate, Salary ,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate)
FROM Employees ORDER BY 1,2;
```

The results from the above query are:

dID	StartDate	Salary	Row_number()
100	1984-08-28	47500.000	1
100	1985-01-01	62000.500	2
100	1985-06-17	57490.000	3
100	1986-06-07	72995.000	4
100	1986-07-01	48023.690	5
...
200	1985-02-03	38500.000	1
200	1985-12-06	54800.000	2
200	1987-02-19	39300.000	3
200	1987-07-10	49500.000	4
...
500	1994-02-27	24903.000	9

Windowing Aggregate Functions

Windowing aggregate functions let you manipulate multiple levels of aggregation in the same query.

For example, you can list all quarters during which expenses are less than the average. You can use aggregate functions, including the simple aggregate functions **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM**, to place results—possibly computed at different levels in the statement—on the same row. This placement provides a means to compare aggregate values with detail rows within a group, avoiding the need for a join or a correlated subquery.

These functions also let you compare nonaggregate values to aggregate values. For example, a salesperson might need to compile a list of all customers who ordered more than the average number of a product in a specified year, or a manager might want to compare an employee's salary against the average salary of the department.

If a query specifies **DISTINCT** in the **SELECT** statement, then the **DISTINCT** operation is applied after the window operator. A window operator is computed after processing the **GROUP BY** clause and before the evaluation of the **SELECT** list items and a query's **ORDER BY** clause.

Windowing aggregate example 1—This query returns a result set, partitioned by year, that shows a list of the products that sold higher-than-average sales.

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
  Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
  CAST(AVG(Sal) OVER(PARTITION BY DepartmentID) AS
  numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
  OVER(PARTITION BY DepartmentID) AS numeric(10,2)) AS
  STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
  Sal > (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

The results from the query:

E_name	Dept	Sal	Average	STD_DEV
Lull	100		87900.00	58736.28
Sheffield	100		87900.00	58736.28
Scott	100	100	96300.00	58736.28
Sterling	200		64900.00	48390.94
Savarino	200		72300.00	48390.94
Kelly	200	200	87500.00	48390.94
Shea	300	300	138948.00	59500.00
Blaikie	400	400	54900.00	43640.67
Morris	400	400	61300.00	43640.67
Evans	400	400	68940.00	43640.67
Martinez	500		55500.80	33752.20

For the year 2000, the average number of orders was 1,787. Four products (700, 601, 600, and 400) sold higher than that amount. In 2001, the average number of orders was 1,048 and 3 products exceeded that amount.

Windowing aggregate example 2—This query returns a result set that shows the employees whose salary is one standard deviation greater than the average salary of their department. Standard deviation is a measure of how much the data varies from the mean.

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
  Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
  CAST(AVG(Sal) OVER(PARTITION BY dept) AS
  numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
  OVER(PARTITION BY dept) AS numeric(10,2)) AS
  STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
  Sal > (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

Every department has at least one employee whose salary significantly deviates from the mean, as shown in these results:

E_name	Dept	Sal	Average	STD_DEV
Lull	100	87900.00	58736.28	16829.59
Sheffield	100	87900.00	58736.28	16829.59
Scott	100	96300.00	58736.28	16829.59
Sterling	200	64900.00	48390.94	13869.59
Savarino	200	72300.00	48390.94	13869.59
Kelly	200	87500.00	48390.94	13869.59
Shea	300	138948.00	59500.00	30752.39
Blaikie	400	54900.00	43640.67	11194.02
Morris	400	61300.00	43640.67	11194.02
Evans	400	68940.00	43640.67	11194.02
Martinez	500	55500.80	33752.20	9084.49

Employee Scott earns \$96,300.00, while the average salary for department 100 is \$58,736.28. The standard deviation for department 100 is 16,829.00, which means that salaries less than \$75,565.88 ($58736.28 + 16829.60 = 75565.88$) fall within one standard deviation of the mean.

Statistical Aggregate Functions

The ANSI SQL/OLAP extensions provide a number of additional aggregate functions that permit statistical analysis of numeric data. This support includes functions to compute variance, standard deviation, correlation, and linear regression.

Standard deviation and variance

The SQL/OLAP general set functions that take one argument include those appearing in bold in this syntax statement:

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
```

	STDDEV		STDDEV_POP		STDDEV_SAMP
	VARIANCE		VARIANCE_POP		VARIANCE_SAMP

- **STDDEV_POP** – computes the population standard deviation of the provided value expression evaluated for each row of the group or partition (if **DISTINCT** is specified, each row that remains after duplicates are eliminated), defined as the square root of the population variance.
- **STDDEV_SAMP** – computes the population standard deviation of the provided value expression evaluated for each row of the group or partition (if **DISTINCT** is specified, each row that remains after duplicates are eliminated), defined as the square root of the sample variance.
- **VAR_POP** – computes the population variance of value expression evaluated for each row of the group or partition (if **DISTINCT** is specified, each row that remains after duplicates are eliminated), defined as the sum of squares of the difference of value expression from the mean of value expression, divided by the number of rows (remaining) in the group or partition.
- **VAR_SAMP** – computes the sample variance of value expression evaluated for each row of the group or partition (if **DISTINCT** is specified, each row that remains after duplicates are eliminated), defined as the sum of squares of the difference of value expression, divided by one less than the number of rows (remaining) in the group or partition.

These functions, including **STDDEV** and **VARIANCE**, are true aggregate functions in that they can compute values for a partition of rows as determined by the query's **ORDER BY** clause. As with other basic aggregate functions such as **MAX** or **MIN**, their computation ignores NULL values in the input. Also, regardless of the domain of the expression being analyzed, all variance and standard deviation computation uses IEEE double-precision floating point. If the input to any variance or standard deviation function is the empty set, then each function returns NULL as its result. If **VAR_SAMP** is computed for a single row, it returns NULL, while **VAR_POP** returns the value 0.

Correlation

The SQL/OLAP function that computes a correlation coefficient is:

- **CORR** – returns the correlation coefficient of a set of number pairs.

You can use the CORR function either as a windowing aggregate function (where you specify a window function type over a window name or specification) or as a simple aggregate function with no **OVER** clause.

Covariance

The SQL/OLAP functions that compute covariances include:

- **COVAR_POP** – returns the population covariance of a set of number pairs.
- **COVAR_SAMP** – returns the sample covariance of a set of number pairs.

The covariance functions eliminate all pairs where expression1 or expression2 has a null value.

You can use the covariance functions either as windowing aggregate functions (where you specify a window function type over a window name or specification) or as simple aggregate functions with no `OVER` clause.

Cumulative distribution

The SQL/OLAP function that calculates the relative position of a single value among a group of rows is **CUME_DIST**.

The window specification must contain an `ORDER_BY` clause.

Composite sort keys are not allowed in the `CUME_DIST` function.

Regression analysis

The regression analysis functions calculate the relationship between an independent variable and a dependent variable using a linear regression equation. The SQL/OLAP linear regression functions include:

- **REGR_AVGX** – computes the average of the independent variable of the regression line.
- **REGR_AVGY** – computes the average of the dependent variable of the regression line.
- **REGR_COUNT** – returns an integer representing the number of nonnull number pairs used to fit the regression line.
- **REGR_INTERCEPT** – computes the y-intercept of the regression line that best fits the dependent and independent variables.
- **REGR_R2** – computes the coefficient of determination (the goodness-of-fit statistic) for the regression line.
- **REGR_SLOPE** – computes the slope of the linear regression line fitted to nonnull pairs.
- **REGR_SXX** – returns the sum of squares of the independent expressions used in a linear regression model. Use this function to evaluate the statistical validity of the regression model.
- **REGR_SXY** – returns the sum of products of the dependent and independent variables. Use this function to evaluate the statistical validity of the regression model.
- **REGR_SYY** – returns values that can evaluate the statistical validity of a regression model.

You can use the regression analysis functions either as windowing aggregate functions (where you specify a window function type over a window name or specification) or as simple aggregate functions with no `OVER` clause.

Weighted OLAP aggregates

The weighted OLAP aggregate functions calculate weighted moving averages:

- **EXP_WEIGHTED_AVG** – calculates an exponentially weighted moving average. Weightings determine the relative importance of each quantity comprising the average. Weights in **EXP_WEIGHTED_AVG** decrease exponentially. Exponential weighting applies more weight to the most recent values and decreases the weight for older values, while still applying some weight

- **WEIGHTED_AVG** – calculates a linearly weighted moving average where weights decrease arithmetically over time. Weights decrease from the highest weight for the most recent data points, down to zero for the oldest data point.

The window specification must contain an **ORDER_BY** clause.

Nonstandard database industry extensions

Non-ANSI SQL/OLAP aggregate function extensions used in the database industry include **FIRST_VALUE**, **MEDIAN**, and **LAST_VALUE**.

- **FIRST_VALUE** – returns the first value from a set of values.
- **MEDIAN** – returns the median from an expression.
- **LAST_VALUE** – returns the last value from a set of values.

The **FIRST_VALUE** and **LAST_VALUE** functions require a window specification. You can use the **MEDIAN** function either as windowing aggregate function (where you specify a window function type over a window name or specification) or as a simple aggregate function with no **OVER** clause.

Inter-Row Functions

The inter-row functions, **LAG** and **LEAD**, provide access to previous or subsequent values in a data series, or to multiple rows in a table.

Inter-row functions also partition simultaneously without a self-join. **LAG** provides access to a row at a given physical offset prior to the **CURRENT ROW** in the table or partition. **LEAD** provides access to a row at a given physical offset after the **CURRENT ROW** in the table or partition.

LAG and **LEAD** syntax is identical. Both functions require an **OVER (ORDER_BY)** window specification. For example:

```
LAG (value_expr) [, offset [, default]] OVER ([PARTITION BY window partition] ORDER BY window ordering)
```

and:

```
LEAD (value_expr) [, offset [, default]] OVER ([PARTITION BY window partition] ORDER BY window ordering)
```

The **PARTITION BY** clause in the **OVER (ORDER_BY)** clause is optional. The **OVER (ORDER_BY)** clause cannot contain a window frame **ROWS/RANGE** specification.

value_expr is a table column or expression that defines the offset data to return from the table. You can define other functions in the *value_expr*, with the exception of analytic functions.

For both functions, specify the target row by entering a physical offset. The *offset* value is the number of rows above or below the current row. Enter a nonnegative numeric data type (entering a negative value generates an error). If you enter 0, SAP Sybase IQ returns the current row.

The optional *default* value defines the value to return if the *offset* value goes beyond the scope of the table. The default value of *default* is **NULL**. The data type of *default* must be implicitly

convertible to the data type of the *value_expr* value, or SAP Sybase IQ generates a conversion error.

LAG example 1—The inter-row functions are useful in financial services applications that perform calculations on data streams, such as stock transactions. This example uses the **LAG** function to calculate the percentage change in the trading price of a particular stock. Consider the following trading data from a fictional table called `stock_trades`:

traded at	symbol	price
2009-07-13 06:07:12	SQL	15.84
2009-07-13 06:07:13	TST	5.75
2009-07-13 06:07:14	TST	5.80
2009-07-13 06:07:15	SQL	15.86
2009-07-13 06:07:16	TST	5.90
2009-07-13 06:07:17	SQL	15.86

Note: The fictional `stock_trades` table is not available in the `iqdemo` database.

The query partitions the trades by stock symbol, orders them by time of trade, and uses the **LAG** function to calculate the percentage increase or decrease in trade price between the current trade and the previous trade:

```
select stock_symbol as 'Stock',
       traded_at    as 'Date/Time of Trade',
       trade_price  as 'Price/Share',
       cast ( ( ( trade_price
                 - (lag(trade_price, 1)
                    over (partition by stock_symbol
                          order by traded_at)))
              / trade_price)
           * 100.0) as numeric(5, 2) )
       as '% Price Change vs Previous Price'
from stock_trades
order by 1, 2
```

The query returns these results:

Stock symbol	Date/Time of Trade	Price/Share	% Price Change_vs Previous Price
SQL	2009-07-13 06:07:12	15.84	NULL
SQL	2009-07-13 06:07:15	15.86	0.13
SQL	2009-07-13 06:07:17	15.86	0.00
TST	2009-07-13 06:07:13	5.75	NULL
TST	2009-07-13 06:07:14	5.80	0.87
TST	2009-07-13 06:07:16	5.90	1.72

The NULL result in the first and fourth output rows indicates that the **LAG** function is out of scope for the first row in each of the two partitions. Since there is no previous row to compare to, SAP Sybase IQ returns NULL as specified by the *default* variable.

Distribution Functions

SQL/OLAP defines several functions that deal with ordered sets.

The two inverse distribution functions are **PERCENTILE_CONT** and **PERCENTILE_DISC**. These analytical functions take a percentile value as the function argument and operate on a group of data specified in the **WITHIN GROUP** clause or operate on the entire data set.

These functions return one value per group. For **PERCENTILE_DISC** (discrete), the data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. For **PERCENTILE_CONT** (continuous), the data type of the results is either numeric, if the **ORDER BY** item in the **WITHIN GROUP** clause is a numeric, or double, if the **ORDER BY** item is an integer or floating point.

The inverse distribution analytical functions require a **WITHIN GROUP (ORDER BY)** clause. For example:

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

The value of *expression1* must be a constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a “wrong argument for percentile” error is returned. If the argument value is less than 0, or greater than 1, then a “data value out of range” error is returned.

The **ORDER BY** clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **WITHIN GROUP** clause and is not an **ORDER BY** for the **SELECT** statement.

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result.

The value *expression2* is a sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Inverse distribution analytical functions are allowed in a subquery, a **HAVING** clause, a view, or a union. The inverse distribution functions can be used anywhere the simple nonanalytical aggregate functions are used. The inverse distribution functions ignore the NULL value in the data set.

PERCENTILE_CONT example—This example uses the **PERCENTILE_CONT** function to determine the 10th percentile value for car sales in a region using the following data set:

sales	region	dealer_name
900	Northeast	Boston

800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin
500	South	Houston
400	South	Austin
300	South	Dallas
200	South	Dover

In the following example query, the **SELECT** statement contains the **PERCENTILE_CONT** function:

```
SELECT region, PERCENTILE_CONT(0.1) WITHIN GROUP
  ( ORDER BY ProductID DESC )
FROM ViewSalesOrdersSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

region	percentile_cont
Canada	601.0
Central	700.0
Eastern	700.0
South	700.0
Western	700.0

PERCENTILE_DISC example—This example uses the **PERCENTILE_DISC** function to determine the 10th percentile value for car sales in a region, using the following data set:

sales	region	dealer_name
900	Northeast	Boston
800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin
500	South	Houston
400	South	Austin
300	South	Dallas
200	South	Dover

In the following query, the **SELECT** statement contains the **PERCENTILE_DISC** function:

```
SELECT region, PERCENTILE_DISC(0.1) WITHIN GROUP
  (ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in each region:

region	percentile_cont
Northeast	900
Northwest	800
South	500

PERCENTILE_CONT Function [Analytical]

Given a percentile, returns the value that corresponds to that percentile. Assumes a continuous distribution data model.

Note: If you are simply trying to compute a percentile, use the **NTILE** function instead, with a value of 100.

Syntax

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression1	A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, a “wrong argument for percentile” error is returned. If the argument value is less than 0 or greater than 1, a “data value out of range” error is returned.
expression2	A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Remarks

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function

PERCENTILE_CONT takes a percentile value as the function argument, and operates on a group of data specified in the **WITHIN GROUP** clause, or operates on the entire data set. The function returns one value per group. If the **GROUP BY** column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its **ORDER**

BY item specified in the **WITHIN GROUP** clause. The data type of the **ORDER BY** expression for **PERCENTILE_CONT** must be numeric.

PERCENTILE_CONT requires a **WITHIN GROUP (ORDER BY)** clause.

The **ORDER BY** clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. For the **PERCENTILE_CONT** function, the data type of this expression must be numeric. This **ORDER BY** clause is used only within the **WITHIN GROUP** clause and is not an **ORDER BY** for the **SELECT**.

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result. The **WITHIN GROUP** clause must contain a single sort item. If the **WITHIN GROUP** clause contains more or less than one sort item, an error is reported.

The **ASC** or **DESC** parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The **PERCENTILE_CONT** function is allowed in a subquery, a **HAVING** clause, a view, or a union. **PERCENTILE_CONT** can be used anywhere the simple nonanalytical aggregate functions are used. The **PERCENTILE_CONT** function ignores the **NULL** value in the data set.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server or SQL Anywhere.

Example

The following example uses the **PERCENTILE_CONT** function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

sales	region	dealer_name
900	Northeast	Boston
800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin
500	South	Houston
400	South	Austin
300	South	Dallas
200	South	Dover

The following **SELECT** statement contains the **PERCENTILE_CONT** function:

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

```
region          percentile_cont
-----          -
Northeast      840
Northwest      740
South          470
```

PERCENTILE_DISC Function [Analytical]

Given a percentile, returns the value that corresponds to that percentile. Assumes a discrete distribution data model.

Note: If you are simply trying to compute a percentile, use the **NTILE** function instead, with a value of 100.

Syntax

```
PERCENTILE_DISC ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression1	A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a “wrong argument for percentile” error is returned. If the argument value is less than 0 or greater than 1, then a “data value out of range” error is returned.
expression2	A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Remarks

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function

PERCENTILE_DISC takes a percentile value as the function argument and operates on a group of data specified in the **WITHIN GROUP** clause or operates on the entire data set. The function returns one value per group. If the **GROUP BY** column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. **PERCENTILE_DISC** supports all data types that can be sorted in SAP Sybase IQ.

PERCENTILE_DISC requires a **WITHIN GROUP (ORDER BY)** clause.

The **ORDER BY** clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **WITHIN GROUP** clause and is not an **ORDER BY** for the **SELECT**.

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result. The **WITHIN GROUP** clause must contain a single sort item. If the **WITHIN GROUP** clause contains more or less than one sort item, an error is reported.

The **ASC** or **DESC** parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The **PERCENTILE_DISC** function is allowed in a subquery, a **HAVING** clause, a view, or a union. **PERCENTILE_DISC** can be used anywhere the simple nonanalytical aggregate functions are used. The **PERCENTILE_DISC** function ignores the **NULL** value in the data set.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server or SQL Anywhere.

Example

The following example uses the **PERCENTILE_DISC** function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

sales	region	dealer_name
900	Northeast	Boston
800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin
500	South	Houston
400	South	Austin
300	South	Dallas
200	South	Dover

The following **SELECT** statement contains the **PERCENTILE_DISC** function:

```
SELECT region, PERCENTILE_DISC(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

region	percentile_cont
Northeast	900
Northwest	800
South	500

Numeric Functions

OLAP numeric functions supported by SAP Sybase IQ include **CEILING** (**CEIL** is an alias), **EXP** (**EXPONENTIAL** is an alias), **FLOOR**, **LN** (**LOG** is an alias), **SQRT**, and **WIDTH_BUCKET**.

```
<numeric value function> ::= =
<natural logarithm>
| <exponential function>
| <power function>
| <square root>
| <floor function>
| <ceiling function>
| <width bucket function>
```

Table 4. Numeric value functions and syntax

Numeric value function	Syntax
Natural logarithm	LN (<i>numeric-expression</i>)
Exponential function	EXP (<i>numeric-expression</i>)
Power function	POWER (<i>numeric-expression1</i> , <i>numeric-expression2</i>)
Square root	SQRT (<i>numeric-expression</i>)
Floor function	FLOOR (<i>numeric-expression</i>)
Ceiling function	CEILING (<i>numeric-expression</i>)
Width bucket function	WIDTH_BUCKET (<i>expression</i> , <i>min_value</i> , <i>max_value</i> , <i>num_buckets</i>)

The semantics of the numeric value functions are:

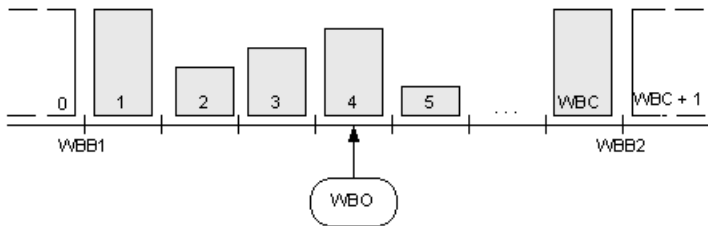
- **LN** – returns the natural logarithm of the argument value. Raises an error condition if the argument value is 0 or negative. LN is a synonym for **LOG**.
- **EXP** – returns the value computed by raising the value of *e* (the base of natural logarithms) to the power specified by the value of the argument.
- **POWER** – returns the value computed by raising the value of the first argument to the power specified by the value of the second argument. If the first argument is 0 and the second is 0, returns one. If the first argument is 0 and the second is positive, returns 0. If the first argument is 0 and the second argument is negative, raises an exception. If the first argument is negative and the second is not an integer, raises an exception.
- **SQRT** – returns the square root of the argument value, defined by syntax transformation to “**POWER** (*expression*, 0.5).”

- **FLOOR** – returns the integer value nearest to positive infinity that is not greater than the value of the argument.
- **CEILING** – returns the integer value nearest to negative infinity that is not less than the value of the argument. CEIL is a synonym for CEILING.

WIDTH_BUCKET function

The **WIDTH_BUCKET** function is somewhat more complicated than the other numeric value functions. It accepts four arguments: “live value,” two range boundaries, and the number of equal-sized (or as nearly so as possible) partitions into which the range indicated by the boundaries is to be divided. **WIDTH_BUCKET** returns a number indicating the partition into which the live value should be placed, based on its value as a percentage of the difference between the higher range boundary and the lower boundary. The first partition is partition number one.

To avoid errors when the live value is outside the range of boundaries, live values that are less than the smaller range boundary are placed into an additional first bucket, bucket zero, and live values that are greater than the larger range boundary are placed into an additional last bucket, bucket N+1.



For example, **WIDTH_BUCKET** (14, 5, 30, 5) returns 2 because:

- $(30-5)/5$ is 5, so the range is divided into 5 partitions, each 5 units wide.
- The first bucket represents values from 0.00% to 19.999 ...%; the second represents values from 20.00% to 39.999 ...%; and the fifth bucket represents values from 80.00% to 100.00%.
- The bucket chosen is determined by computing $(5*(14-5)/(30-5)) + 1$ — one more than the number of buckets times the ratio of the offset of the specified value from the lower value to the range of possible values, which is $(5*0/25) + 1$, which is 2.8. This value is the range of values for bucket number 2 (2.0 through 2.999 ...), so bucket number 2 is chosen.

WIDTH_BUCKET example

The following example creates a ten-bucket histogram on the `credit_limit` column for customers in Massachusetts in the sample table and returns the bucket number (“Credit Group”) for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

Note: This example is for illustration purposes only and was not generated using the iqdemo database.

```
SELECT customer_id, cust_last_name, credit_limit,
       WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
       Group"
FROM customers WHERE territory = 'MA'
ORDER BY "Credit Group";
```

CUSTOMER_ID	CUST_LAST_NAME	CREDIT_LIMIT	Credit Group
825	Dreyfuss	500	1
826	Barkin	500	1
853	Palin	400	1
827	Siegel	500	1
843	Oates	700	2
844	Julius	700	2
835	Eastwood	1200	3
840	Elliott	1400	3
842	Stern	1400	3
841	Boyer	1400	3
837	Stanton	1200	3
836	Berenger	1200	3
848	Olmos	1800	4
847	Streep	5000	11

When the bounds are reversed, the buckets are open-closed intervals. For example: **WIDTH_BUCKET** (*credit_limit*, 5000, 0, 5). In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

BIT_LENGTH Function [String]

Returns an unsigned 64-bit value containing the bit length of the column parameter.

Syntax

```
BIT_LENGTH ( column-name )
```

Parameters

Parameter	Description
column-name	The name of a column

Returns

INT

Remarks

The return value of a NULL argument is NULL.

The **BIT_LENGTH** function supports all SAP Sybase IQ data types.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Function Support in Unstructured Data Analytics*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by SQL Anywhere or Adaptive Server.

CEIL Function [Numeric]

Returns the smallest integer greater than or equal to the specified expression.

CEIL is as synonym for **CEILING**.

Syntax

```
CEIL ( numeric-expression )
```

Parameters

Parameters	Description
expression	A column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, CEIL generates an error. The return value has the same data type as the value supplied.

Remarks

For a given expression, the **CEIL** function takes one argument. For example, **CEIL (-123.45)** returns -123. **CEIL (123.45)** returns 124.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

CEILING Function [Numeric]

Returns the ceiling (smallest integer not less than) of a number.

CEIL is as synonym for **CEILING**.

Syntax

```
CEILING ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number whose ceiling is to be calculated.

Returns

DOUBLE

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server.

Example

The following statement returns the value 60.00000:

```
SELECT CEILING( 59.84567 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT CEILING( 123 ) FROM iq_dummy
```

The following statement returns the value 124.00:

```
SELECT CEILING( 123.45 ) FROM iq_dummy
```

The following statement returns the value -123.00:

```
SELECT CEILING( -123.45 ) FROM iq_dummy
```

EXP Function [Numeric]

Returns the exponential function, e to the power of a number.

Syntax

```
EXP ( numeric-expression )
```

Parameters

Table 5. Parameters

Parameter	Description
numeric-expression	The exponent.

Returns

DOUBLE

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

Example

The following statement returns the value 3269017.3724721107:

```
SELECT EXP ( 15 ) FROM iq_dummy
```

FLOOR Function [Numeric]

Returns the floor of (largest integer not greater than) a number.

Syntax

```
FLOOR ( numeric-expression )
```

*Parameters***Table 6. Parameters**

Parameter	Description
numeric-expression	The number, usually a float.

Returns

DOUBLE

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

Example

The following statement returns the value 123.00:

```
SELECT FLOOR ( 123 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```

The following statement returns the value -124.00.

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

LN Function [Numeric]

Returns the natural logarithm of the specified expression.

Syntax

```
LN ( numeric-expression )
```

Parameters

Parameter	Description
expression	Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the LN function generates an error. The return value is of DOUBLE data type.

Remarks

LN takes one argument. For example, **LN** (20) returns 2.995732.

The **LN** function is an alias of the **LOG** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use the **LOG** function instead.

POWER Function [Numeric]

Calculates one number raised to the power of another.

Syntax

```
POWER ( numeric-expression1, numeric-expression2 )
```

Parameters

Parameter	Description
numeric-expression1	The base.
numeric-expression2	The exponent.

Returns

DOUBLE

Remarks

Raises *numeric-expression1* to the power *numeric-expression2*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

Example

The following statement returns the value 64:

```
SELECT Power( 2, 6 ) FROM iq_dummy
```

SQRT Function [Numeric]

Returns the square root of a number.

Syntax

```
SQRT ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number for which the square root is to be calculated.

Returns

DOUBLE

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

Example

The following statement returns the value 3:

```
SELECT SQRT( 9 ) FROM iq_dummy
```

WIDTH_BUCKET Function [Numerical]

For a given expression, the **WIDTH_BUCKET** function returns the bucket number that the result of this expression will be assigned after it is evaluated.

Syntax

```
WIDTH_BUCKET ( expression, min_value, max_value, num_buckets )
```

Parameters

Parameter	Description
expression	The expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If <i>expr</i> evaluates to null, then the expression returns null.
min_value	An expression that resolves to the end points of the acceptable range for <i>expr</i> . Must also evaluate to numeric or datetime values and cannot evaluate to null.
max_value	An expression that resolves to the end points of the acceptable range for <i>expr</i> . Must also evaluate to numeric or datetime values and cannot evaluate to null.
num_buckets	Is an expression that resolves to a constant indicating the number of buckets. This expression must evaluate to a positive integer.

Remarks

You can generate equiwidth histograms with the **WIDTH_BUCKET** function. Equiwidth histograms divide data sets into buckets whose interval size (highest value to lowest value) is equal. The number of rows held by each bucket will vary. A related function, **NTILE**, creates equiheigh buckets.

Equiwidth histograms can be generated only for numeric, date or datetime data types; therefore, the first three parameters should be all numeric expressions or all date expressions. Other types of expressions are not allowed. If the first parameter is NULL, the result is NULL. If the second or the third parameter is NULL, an error message is returned, as a NULL value cannot denote any end point (or any point) for a range in a date or numeric value dimension. The last parameter (number of buckets) should be a numeric expression that evaluates to a positive integer value; 0, NULL, or a negative value will result in an error.

Buckets are numbered from 0 to (n+1). Bucket 0 holds the count of values less than the minimum. Bucket(n+1) holds the count of values greater than or equal to the maximum specified value.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported by Adaptive Server Enterprise.

Example

The following example creates a ten-bucket histogram on the `credit_limit` column for customers in Massachusetts in the sample table and returns the bucket number (“Credit Group”) for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

```
select EmployeeID, Surname, Salary, WIDTH_BUCKET(Salary, 29000,
60000, 4) "Wages" from Employees where State = 'FL' order by "Wages"
```

EMPLOYEEID	SURNAME	SALARY	Wages
888	Charlton	28300.000	0
1390	Litton	58930.000	4
207	Francis	53870.000	4
266	Gowda	59840.000	4
445	Lull	87900.000	5
1021	Sterling	64900.000	5
902	Kelly	87500.000	5
1576	Evans	68940.000	5

When the bounds are reversed, the buckets are open-closed intervals. For example: **WIDTH_BUCKET** (*credit_limit*, 5000, 0, 5). In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

OLAP Rules and Restrictions

The following provides an overview for the rules and restrictions that govern OLAP functionality.

OLAP Functions Can be Used

SAP Sybase IQ provides SQL OLAP functions with rules, restrictions and limitations.

- In the **SELECT** list
- In expressions
- As arguments of scalar functions
- In the final **ORDER BY** clause (by using aliases or positional references to OLAP functions elsewhere in the query)

OLAP Functions Cannot be Used

OLAP functions cannot be used under these conditions:

- In subqueries.
- In the search condition of a **WHERE** clause.
- As arguments for **SET** (aggregate) functions. For example, the following expression is invalid:

```
SUM(RANK() OVER(ORDER BY dollars))
```

- A windowed aggregate cannot be an argument to another unless the inner one was generated within a view or derived table. The same applies to ranking functions.
- Window aggregate and **RANK** functions are not allowed in a **HAVING** clause.
- Window aggregate functions should not specify **DISTINCT**.
- Window function cannot be nested inside of other window functions.
- Inverse distribution functions are not supported with the **OVER** clause.
- Outer references are not allowed in a window definition clause.
- Correlation references are allowed within OLAP functions, but correlated column aliases are not allowed.

Columns referenced by an OLAP function must be grouping columns or aggregate functions from the same query block in which the OLAP function and the **GROUP BY** clause appear. OLAP processing occurs after the grouping and aggregation operations and before the final **ORDER BY** clause is applied; therefore, it must be possible to derive the OLAP expressions from those intermediate results. If there is no **GROUP BY** clause in a query block, OLAP functions can reference other columns in the select list.

SAP Sybase IQ Limitations

The SAP Sybase IQ limitations with SQL OLAP functions are:

- User-defined functions in a window frame definition are not supported.
- The constants used in a window frame definition must be unsigned numeric value and should not exceed the value of maximum `BIG INT 263-1`.
- Window aggregate functions and **RANK** functions cannot be used in **DELETE** and **UPDATE** statements.
- Window aggregate and **RANK** functions are not allowed in subqueries.
- `CUME_DIST` is currently not supported.
- Grouping sets are currently not supported.
- Correlation and linear regression functions are currently not supported.

Additional OLAP Examples

This section provides additional examples using the OLAP functions.

Both start and end points of a window may vary as intermediate result rows are processed. For example, computing a cumulative sum involves a window with the start point fixed at the first row of each partition and an end point that slides along the rows of the partition to include the current row.

As another example, both the start and end points of the window can be variable yet define a constant number of rows for the entire partition. Such a construction lets users compose queries that compute moving averages; for example, a SQL query that returns a moving three-day average stock price.

Example: Window Functions in Queries

This query lists all products shipped in July and August 2005 and the cumulative shipped quantity by shipping date:

```
SELECT p.id, p.description, s.quantity, s.shipdate,
SUM(s.quantity) OVER (PARTITION BY productid ORDER BY s.shipdate rows
between unbounded preceding and current row)FROM SalesOrderItems s
JOIN Products p on(s.ProductID =p.id) WHERE s.ShipDate BETWEEN
'2001-05-01' and '2001-08-31' AND s.quantity > 40 ORDER BY p.id;
```

ID	description	quantity	ship_date	sum quantity
302	Crew Neck	60	2001-07-02	60
400	Cotton Cap	60	2001-05-26	60
400	Cotton Cap	48	2001-07-05	108
401	Wool cap	48	2001-06-02	48
401	Wool cap	60	2001-06-30	108
401	Wool cap	48	2001-07-09	156
500	Cloth Visor	48	2001-06-21	48
501	Plastic Visor	60	2001-05-03	60
501	Plastic Visor	48	2001-05-18	108
501	Plastic Visor	48	2001-05-25	156
501	Plastic Visor	60	2001-07-07	216
601	Zippered Sweatshirt	60	2001-07-19	60
700	Cotton Shorts	72	2001-05-18	72
700	Cotton Shorts	48	2001-05-31	120

In this example, the computation of the **SUM** window function occurs after the join of the two tables and the application of the query's **WHERE** clause. The query uses an inline window specification that specifies that the input rows from the join is processed as follows:

1. Partition (group) the input rows based on the value of the `prod_id` attribute.
2. Within each partition, sort the rows by the `ship_date` attribute.
3. For each row in the partition, evaluate the **SUM()** function over the quantity attribute, using a sliding window consisting of the first (sorted) row of each partition, up to and including the current row.

An alternative construction for the query is to specify the window separate from the functions that use it. This is useful when more than one window function is specified that are based on the same window. In the case of the query using window functions, a construction that uses the window clause (declaring a window identified by cumulative) is as follows:

```
SELECT p.id, p.description, s.quantity, s.shipdate, SUM(s.quantity)
OVER(cumulative ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW )
cumulative FROM SalesOrderItems s JOIN Products p On (s.ProductID
=p.id)WHERE s.shipdate BETWEEN '2001-07-01' and '2001-08-31'Window
cumulative as (PARTITION BY s.productid ORDER BY s.shipdate)ORDER BY
p.id;
```

The window clause appears before the **ORDER BY** clause in the query specification. When using a window clause, the following restrictions apply:

Appendix: Using OLAP

- The inline window specification cannot contain a **PARTITION BY** clause.
- The window specified within the window clause cannot contain a window frame clause.

```
<WINDOW FRAME CLAUSE> ::=  
  <WINDOW FRAME UNIT>  
  <WINDOW FRAME EXTENT>
```

- Either the inline window specification, or the window specification specified in the window clause, can contain a window order clause, but not both.

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

Example: Window With Multiple Functions

This query defines a single (named) window and computes multiple function results over it:

```
SELECT p.ID, p.Description, s.quantity, s.ShipDate, SUM(s.Quantity)  
  OVER wsl, MIN(s.quantity) OVER wsl  
FROM SalesOrderItems s  
JOIN Products p ON (s.ProductID =p.ID)  
  WHERE s.ShipDate BETWEEN '2000-01-09' AND'2000-01-17'  
  AND s.Quantity > 40 window wsl  
  AS(PARTITION BY productid  
  ORDER BY shipdate rows between unbounded preceding and current row)  
ORDER BY p.id;
```

ID	Description	quantity	shipDate	SUM	MIN
400	Cotton Cap	48	2000-01-09	48	48
401	Wool cap	48	2000-01-09	48	48
500	Cloth Visor	60	2000-01-14	60	60
500	Cloth Visor	60	2000-01-15	120	60
501	Plastic Visor	60	2000-01-14	60	60

Example: Calculate Cumulative Sum

This query calculates a cumulative sum of salary per department and **ORDER BY** start_date.

```
SELECT dept_id, start_date, name, salary,  
  SUM(salary) OVER (PARTITION BY dept_id ORDER BY  
  start_date ROWS BETWEEN UNBOUNDED PRECEDING AND  
  CURRENT ROW)  
FROM emp1  
ORDER BY dept_id, start_date;
```

DepartmentID	start_date	name	salary	sum(salary)
100	1996-01-01	Anna	18000	18000
100	1997-01-01	Mike	28000	46000
100	1998-01-01	Scott	29000	75000
100	1998-02-01	Antonia	22000	97000
100	1998-03-12	Adam	25000	122000
100	1998-12-01	Amy	18000	140000
200	1998-01-01	Jeff	18000	18000
200	1998-01-20	Tim	29000	47000

200	1998-02-01	Jim	22000	69000
200	1999-01-10	Tom	28000	97000
300	1998-03-12	Sandy	55000	55000
300	1998-12-01	Lisa	38000	93000
300	1999-01-10	Peter	48000	141000

Example: Calculate Moving Average

This query generates the moving average of sales in three consecutive months. The size of the window frame is three rows: two preceding rows plus the current row. The window slides from the beginning to the end of the partition.

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	avg(sales)
-----	-----	-----	-----
10	1	100	100.00
10	2	120	110.00
10	3	100	106.66
10	4	130	116.66
10	5	120	116.66
10	6	110	120.00
20	1	20	20.00
20	2	30	25.00
20	3	25	25.00
20	4	30	28.33
20	5	31	28.66
20	6	20	27.00
30	1	10	10.00
30	2	11	10.50
30	3	12	11.00
30	4	1	8.00

Example: ORDER BY Results

In this example, the top **ORDER BY** clause of a query is applied to the final results of a window function. The **ORDER BY** in a window clause is applied to the input data of a window function.

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id desc, month_num;
```

prod_id	month_num	sales	avg(sales)
-----	-----	-----	-----
30	1	10	10.00
30	2	11	10.50
30	3	12	11.00
30	4	1	8.00
20	1	20	20.00
20	2	30	25.00

20	3	25	25.00
20	4	30	28.33
20	5	31	28.66
20	6	20	27.00
10	1	100	100.00
10	2	120	110.00
10	3	100	106.66
10	4	130	116.66
10	5	120	116.66
10	6	110	120.00

Example: Multiple Aggregate Functions in a Query

This example calculates aggregate values against different windows in a query.

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (WS1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS
  CAvg, SUM(sales) OVER(WS1 ROWS BETWEEN UNBOUNDED
  PRECEDING AND CURRENT ROW) AS CSum
FROM sale WHERE rep_id = 1 WINDOW WS1 AS (PARTITION BY
  prod_id
  ORDER BY month_num)
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	CAvg	CSum
10	1	100	110.00	100
10	2	120	106.66	220
10	3	100	116.66	320
10	4	130	116.66	450
10	5	120	120.00	570
10	6	110	115.00	680
20	1	20	25.00	20
20	2	30	25.00	50
20	3	25	28.33	75
20	4	30	28.66	105
20	5	31	27.00	136
20	6	20	25.50	156
30	1	10	10.50	10
30	2	11	11.00	21
30	3	12	8.00	33
30	4	1	6.50	34

Example: Window Frame Comparing ROWS and RANGE

This query compares ROWS and RANGE. The data contain duplicate ROWS per the ORDER BY clause.

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (ws1 RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) AS
  Range_sum, SUM(sales) OVER
  (ws1 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
  Row_sum
FROM sale window ws1 AS (PARTITION BY prod_id ORDER BY
  month_num)
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	Range_sum	Row_sum
10	1	100	250	100
10	1	150	250	250
10	2	120	370	370
10	3	100	470	370
10	4	130	350	350
10	5	120	381	350
10	5	31	381	281
10	6	110	391	261
20	1	20	20	20
20	2	30	50	50
20	3	25	75	75
20	4	30	85	85
20	5	31	86	86
20	6	20	81	81
30	1	10	10	10
30	2	11	21	21
30	3	12	33	33
30	4	1	25	24
30	4	1	25	14

Example: Window Frame Excludes Current Row

In this example, you can define the window frame to exclude the current row. The query calculates the sum over four rows, excluding the current row.

```
SELECT prod_id, month_num, sales, sum(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEN 6 PRECEDING AND 2 PRECEDING)
FROM sale
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	sum(sales)
10	1	100	(NULL)
10	1	150	(NULL)
10	2	120	(NULL)
10	3	100	250
10	4	130	370
10	5	120	470
10	5	31	470
10	6	110	600
20	1	20	(NULL)
20	2	30	(NULL)
20	3	25	20
20	4	30	50
20	5	31	75
20	6	20	105
30	1	10	(NULL)
30	2	11	(NULL)
30	3	12	10
30	4	1	21
30	4	1	21

Example: Window Frame for RANGE

This query illustrates the RANGE window frame. The number of rows used in the summation is variable.

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEN 1 FOLLOWING AND 3 FOLLOWING)
FROM sale
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	sum(sales)
10	1	100	350
10	1	150	350
10	2	120	381
10	3	100	391
10	4	130	261
10	5	120	110
10	5	31	110
10	6	110	(NULL)
20	1	20	85
20	2	30	86
20	3	25	81
20	4	30	51
20	5	31	20
20	6	20	(NULL)
30	1	10	25
30	2	11	14
30	3	12	2
30	4	1	(NULL)
30	4	1	(NULL)

Example: Unbounded Preceding and Unbounded Following

In this example, the window frame can include all rows in the partition. The query calculates max(sales) sale over the entire partition (no duplicate rows in a month).

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	SUM(sales)
10	1	100	680
10	2	120	680
10	3	100	680
10	4	130	680
10	5	120	680
10	6	110	680
20	1	20	156
20	2	30	156
20	3	25	156

20	4	30	156
20	5	31	156
20	6	20	156
30	1	10	34
30	2	11	34
30	3	12	34
30	4	1	34

The query in this example is equivalent to:

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id )
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

Example: Default Window Frame for RANGE

This query illustrates the default window frame for RANGE:

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num)
FROM sale
ORDER BY prod_id, month_num;
```

prod_id	month_num	sales	SUM(sales)
-----	-----	-----	-----
10	1	100	250
10	1	150	250
10	2	120	370
10	3	100	470
10	4	130	600
10	5	120	751
10	5	31	751
10	6	110	861
20	1	20	20
20	2	30	50
20	3	25	75
20	4	30	105
20	5	31	136
20	6	20	156
30	1	10	10
30	2	11	21
30	3	12	33
30	4	1	35
30	4	1	35

The query in this example is equivalent to:

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM sale
ORDER BY prod_id, month_num;
```

BNF Grammar for OLAP Functions

The Backus-Naur Form grammar outlines the specific syntactic support for the various ANSI SQL analytic functions, many of which are implemented in SAP Sybase IQ.

Grammar Rule 1

```
<SELECT LIST EXPRESSION> ::=
  <EXPRESSION>
  | <GROUP BY EXPRESSION>
  | <AGGREGATE FUNCTION>
  | <GROUPING FUNCTION>
  | <TABLE COLUMN>
  | <WINDOWED TABLE FUNCTION>
```

Grammar Rule 2

```
<QUERY SPECIFICATION> ::=
  <FROM CLAUSE>
  [ <WHERE CLAUSE> ]
  [ <GROUP BY CLAUSE> ]
  [ <HAVING CLAUSE> ]
  [ <WINDOW CLAUSE> ]
  [ <ORDER BY CLAUSE> ]
```

Grammar Rule 3

```
<ORDER BY CLAUSE> ::= <ORDER SPECIFICATION>
```

Grammar Rule 4

```
<GROUPING FUNCTION> ::=
  GROUPING <LEFT PAREN> <GROUP BY EXPRESSION>
  <RIGHT PAREN>
```

Grammar Rule 5

```
<WINDOWED TABLE FUNCTION> ::=
  <WINDOWED TABLE FUNCTION TYPE> OVER <WINDOW NAME OR
  SPECIFICATION>
```

Grammar Rule 6

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
  | ROW NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

Grammar Rule 7

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE RANK | PERCENT RANK | CUME_DIST
```


Grammar Rule 8

```
<WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

Grammar Rule 9

```
<AGGREGATE FUNCTION> ::=
  <DISTINCT AGGREGATE FUNCTION>
  | <SIMPLE AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

Grammar Rule 10

```
<DISTINCT AGGREGATE FUNCTION> ::=
  <BASIC AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DISTINCT> <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> DISTINCT <EXPRESSION>
  [ <COMMA> <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

Grammar Rule 11

```
<BASIC AGGREGATE FUNCTION TYPE> ::=
  SUM | MAX | MIN | AVG | COUNT
```

Grammar Rule 12

```
<SIMPLE AGGREGATE FUNCTION> ::=
  <SIMPLE AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> <EXPRESSION> [ <COMMA>
  <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

Grammar Rule 13

```
<SIMPLE AGGREGATE FUNCTION TYPE> ::= <SIMPLE WINDOW AGGREGATE
FUNCTION TYPE>
```

Grammar Rule 14

```
<SIMPLE WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
  | GROUPING FUNCTION
```

Grammar Rule 15

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

Grammar Rule 16

```
<STATISTICAL AGGREGATE FUNCTION> ::=  
  <STATISTICAL AGGREGATE FUNCTION TYPE> <LEFT PAREN>  
  <DEPENDENT EXPRESSION> <COMMA> <INDEPENDENT  
  EXPRESSION> <RIGHT PAREN>
```

Grammar Rule 17

```
<STATISTICAL AGGREGATE FUNCTION TYPE> ::=  
  CORR | COVAR_POP | COVAR_SAMP | REGR_R2 |  
  REGR_INTERCEPT | REGR_COUNT | REGR_SLOPE |  
  REGR_SXX | REGR_SXY | REGR_SYY | REGR_AVGY |  
  REGR_AVGX
```

Grammar Rule 18

```
<WINDOW NAME OR SPECIFICATION> ::=  
  <WINDOW NAME> | <IN-LINE WINDOW SPECIFICATION>
```

Grammar Rule 19

```
<WINDOW NAME> ::= <IDENTIFIER>
```

Grammar Rule 20

```
<IN-LINE WINDOW SPECIFICATION> ::= <WINDOW SPECIFICATION>
```

Grammar Rule 21

```
<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>
```

Grammar Rule 22

```
<WINDOW DEFINITION LIST> ::=  
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>  
  } . . . ]
```

Grammar Rule 23

```
<WINDOW DEFINITION> ::=  
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

Grammar Rule 24

```
<NEW WINDOW NAME> ::= <WINDOW NAME>
```

Grammar Rule 25

```
<WINDOW SPECIFICATION> ::=  
  <LEFT PAREN> <WINDOW SPECIFICATION> <DETAILS> <RIGHT  
  PAREN>
```

Grammar Rule 26

```
<WINDOW SPECIFICATION DETAILS> ::=
[ <EXISTING WINDOW NAME> ]
[ <WINDOW PARTITION CLAUSE> ]
[ <WINDOW ORDER CLAUSE> ]
[ <WINDOW FRAME CLAUSE> ]
```

Grammar Rule 27

```
<EXISTING WINDOW NAME> ::= <WINDOW NAME>
```

Grammar Rule 28

```
<WINDOW PARTITION CLAUSE> ::=
PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

Grammar Rule 29

```
<WINDOW PARTITION EXPRESSION LIST> ::=
<WINDOW PARTITION EXPRESSION>
[ { <COMMA> <WINDOW PARTITION EXPRESSION> } . . . ]
```

Grammar Rule 30

```
<WINDOW PARTITION EXPRESSION> ::= <EXPRESSION>
```

Grammar Rule 31

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

Grammar Rule 32

```
<WINDOW FRAME CLAUSE> ::=
<WINDOW FRAME UNIT>
<WINDOW FRAME EXTENT>
```

Grammar Rule 33

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

Grammar Rule 34

```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

Grammar Rule 35

```
<WINDOW FRAME START> ::=
UNBOUNDED PRECEDING
| <WINDOW FRAME PRECEDING>
| CURRENT ROW
```

Grammar Rule 36

```
<WINDOW FRAME PRECEDING> ::= <UNSIGNED VALUE SPECIFICATION>  
PRECEDING
```

Grammar Rule 37

```
<WINDOW FRAME BETWEEN> ::=  
  BETWEEN <WINDOW FRAME BOUND 1> AND <WINDOW FRAME  
  BOUND 2>
```

Grammar Rule 38

```
<WINDOW FRAME BOUND 1> ::= <WINDOW FRAME BOUND>
```

Grammar Rule 39

```
<WINDOW FRAME BOUND 2> ::= <WINDOW FRAME BOUND>
```

Grammar Rule 40

```
<WINDOW FRAME BOUND> ::=  
  <WINDOW FRAME START>  
  | UNBOUNDED FOLLOWING  
  | <WINDOW FRAME FOLLOWING>
```

Grammar Rule 41

```
<WINDOW FRAME FOLLOWING> ::= <UNSIGNED VALUE SPECIFICATION>  
FOLLOWING
```

Grammar Rule 42

```
<GROUP BY EXPRESSION> ::= <EXPRESSION>
```

Grammar Rule 43

```
<SIMPLE GROUP BY TERM> ::=  
  <GROUP BY EXPRESSION>  
  | <LEFT PAREN> <GROUP BY EXPRESSION> <RIGHT PAREN>  
  | <LEFT PAREN> <RIGHT PAREN>
```

Grammar Rule 44

```
<SIMPLE GROUP BY TERM LIST> ::=  
  <SIMPLE GROUP BY TERM> [ { <COMMA> <SIMPLE GROUP BY  
  TERM> } . . . ]
```

Grammar Rule 45

```
<COMPOSITE GROUP BY TERM> ::=  
  <LEFT PAREN> <SIMPLE GROUP BY TERM>  
  [ { <COMMA> <SIMPLE GROUP BY TERM> } . . . ]  
  <RIGHT PAREN>
```

Grammar Rule 46

```
<ROLLUP TERM> ::= ROLLUP <COMPOSITE GROUP BY TERM>
```

Grammar Rule 47

```
<CUBE TERM> ::= CUBE <COMPOSITE GROUP BY TERM>
```

Grammar Rule 48

```
<GROUP BY TERM> ::=
  <SIMPLE GROUP BY TERM>
  | <COMPOSITE GROUP BY TERM>
  | <ROLLUP TERM>
  | <CUBE TERM>
```

Grammar Rule 49

```
<GROUP BY TERM LIST> ::=
  <GROUP BY TERM> [ { <COMMA> <GROUP BY TERM> } ... ]
```

Grammar Rule 50

```
<GROUP BY CLAUSE> ::= GROUP BY <GROUPING SPECIFICATION>
```

Grammar Rule 51

```
<GROUPING SPECIFICATION> ::=
  <GROUP BY TERM LIST>
  | <SIMPLE GROUP BY TERM LIST> WITH ROLLUP
  | <SIMPLE GROUP BY TERM LIST> WITH CUBE
```

Grammar Rule 52

Not supported.

Grammar Rule 53

```
<ORDER SPECIFICATION> ::= ORDER BY <SORT SPECIFICATION LIST>
  <SORT SPECIFICATION LIST> ::= <SORT SPECIFICATION>
  [ { <COMMA> <SORT SPECIFICATION> } . . . ]
  <SORT SPECIFICATION> ::= <SORT KEY>
  [ <ORDERING SPECIFICATION> ] [ <NULL ORDERING> ]
  <SORT KEY> ::= <VALUE EXPRESSION>
  <ORDERING SPECIFICATION> ::= ASC | DESC
  <NULL ORDERING> := NULLS FIRST | NULLS LAST
```


Appendix: Accessing Remote Data

SAP Sybase IQ can access data located on separate servers, both SAP Sybase and non-SAP Sybase, as if the data were stored on the local server.

You can use this feature to migrate data into an SAP Sybase IQ database or to query data across databases.

SAP Sybase IQ and Remote Data

SAP Sybase IQ remote data access gives you access to data in other data sources. You can use this feature to migrate data into a SQL Anywhere database or query data across databases.

Characteristics of Sybase Open Client and jConnect connections

When SAP Sybase IQ is serving applications over TDS, it automatically sets relevant database options to values compatible with Adaptive Server default behavior. These options are set temporarily, for the duration of the connection only. The client application can override them at any time.

Default settings

The database options set on connection using TDS include:

Option	Set to
allow_nulls_by_default	Off
ansinull	Off
chained	Off
close_on_endtrans	Off
date_format	YYYY-MM-DD
date_order	MDY
escape_character	Off
isolation_level	1
on_tsq_l_error	Continue
quoted_identifier	Off
time_format	HH:NN:SS.SSS

Option	Set to
timestamp_format	YYYY-MM-DD HH:NN:SS.SSS
tsql_variables	On

How the startup options are set

The default database options are set for TDS connections using a system procedure named `sp_tsql_environment`. This procedure sets the following options:

```
SET TEMPORARY OPTION allow_nulls_by_default='Off';
SET TEMPORARY OPTION ansinull='Off';
SET TEMPORARY OPTION chained='Off';
SET TEMPORARY OPTION close_on_endtrans='Off';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='Off';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION on_tsql_error='Continue';
SET TEMPORARY OPTION quoted_identifier='Off';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION tsql_variables='On';
```

Note: Do not alter the `sp_tsql_environment` procedure. It is for system use only.

The procedure sets options only for connections that use the TDS communications protocol. This includes Sybase Open Client and JDBC connections using jConnect. Other connections (ODBC and embedded SQL) have the default settings for the database.

Although SAP Sybase IQ allows longer user names and passwords, TDS client names and passwords cannot exceed 30 bytes. If your password or user ID is longer than 30 bytes, attempts to connect over TDS (for example, using jConnect) return an invalid user or password error.

Changing the option settings for TDS connections

When SAP Sybase IQ is serving applications over TDS, it automatically sets relevant database options to values compatible with Adaptive Server default behavior. You can change the options for TDS connections at any time.

Changing the option settings for TDS connections

When SAP Sybase IQ is serving applications over TDS, it automatically sets relevant database options to values compatible with Adaptive Server default behavior. You can change the options for TDS connections at any time.

1. Create a procedure that sets the database options you want.
2. Set the `login_procedure` option to the name of the new procedure.

Future connections use the procedure. You can configure the procedure differently for different user IDs.

Requirements for Accessing Remote Data

There are several basic elements required to access remote data.

Remote table mappings

SAP Sybase IQ presents tables to a client application as if all the data in the tables were stored in the database to which the application is connected. Internally, when a query involving remote tables is executed, the storage location is determined, and the remote location is accessed so that data can be retrieved.

To access data in a remote table, you must set up the following.

1. You must define the remote server where the remote data is located. This includes the class of server and location of the remote server. The CREATE SERVER statement is used to do this.
2. You must define remote server user login information if the credentials required to access the database on the remote server are different from the database to which you are connected. The CREATE EXTERNLOGIN statement is used to do this.
3. You must create a proxy table definition. This specifies the mapping of a local proxy table to a remote table. This includes the server where the remote table is located, the database name, owner name, table name, and column names of the remote table. The CREATE EXISTING TABLE statement is used to do this. Also, the CREATE TABLE statement can be used to create new tables at the remote server.

To manage remote server definitions, external logins, and proxy table mappings, you can use a tool such as Interactive SQL to execute SQL statements.

Warning! Some remote servers, such as Microsoft Access, Microsoft SQL Server, and Sybase Adaptive Server Enterprise do not preserve cursors across COMMITs and ROLLBACKs. However, you can still use Interactive SQL to view and edit the data in these proxy tables as long as autocommit is turned off (this is the default behavior in Interactive SQL). Other RDBMSs, including Oracle Database, IBM DB2, and SAP Sybase IQ do not have this limitation.

Server classes for remote data access

The server class you specify in the CREATE SERVER statement determines the behavior of a remote connection. The server classes give SAP Sybase IQ detailed server capability information. SAP Sybase IQ formats SQL statements specific to a server's capabilities.

All server classes are ODBC-based. Each server class has a set of unique characteristics that you need to know to configure the server for remote data access. You should refer to information generic to the server class category and also to the information specific to the individual server class.

The server classes include:

SAODBC
ULODBC
ADSODBC
ASEODBC
DB2ODBC
HANAODBC
IQODBC
MSACCESSODBC
MSSODBC
MYSQLODBC
ODBC
ORAODBC

Note: When using remote data access, if you use an ODBC driver that does not support Unicode, then character set conversion is not performed on data coming from that ODBC driver.

ODBC external server definitions

The most common way of defining an ODBC-based remote server is to base it on an ODBC data source. To do this, you can create a data source using the ODBC Data Source Administrator.

Once you have defined the data source, the USING clause in the CREATE SERVER statement should refer to the ODBC Data Source Name (DSN).

For example, to configure an IBM DB2 server named mydb2 whose data source name is also mydb2, use:

```
CREATE SERVER mydb2
CLASS 'DB2ODBC'
USING 'mydb2';
```

The driver used must match the bitness of the database server.

On Windows, you must also define a System Data Source Name (System DSN) with a bitness matching the database server. For example, use the 32-bit ODBC Data Source Administrator to create a 32-bit System DSN. A User DSN does not have bitness.

Using connection strings instead of data sources

An alternative, which avoids using data source names, is to supply a connection string in the USING clause of the CREATE SERVER statement. To do this, you must know the connection parameters for the ODBC driver you are using. For example, a connection to an SAP Sybase IQ database server may be as follows:

```
CREATE SERVER TestSA
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=myhost;Server=TestSA;DBN=sample';
```

This defines a connection to a database server named TestSA, running on a computer called myhost, and a database named sample using the TCP/IP protocol.

USING clause in the CREATE SERVER statement

You must issue a separate CREATE SERVER statement for each remote SAP Sybase IQ database you intend to access. For example, if an SAP Sybase IQ server named TestSA is running on the computer Banana and owns three databases (db1, db2, db3), you would set up the remote servers similar to this:

```
CREATE SERVER TestSAdb1
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db1';

CREATE SERVER TestSAdb2
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db2';

CREATE SERVER TestSAdb3
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db3';
```

If you do not specify a database name, the remote connection uses the remote SAP Sybase IQ server default database.

Server class SAODBC

A remote server with server class SAODBC is an SAP Sybase IQ database server. No special requirements exist for the configuration of an SAP Sybase IQ data source.

To access SAP Sybase IQ database servers that support multiple databases, create an ODBC data source name defining a connection to each database. Execute a CREATE SERVER statement for each of these ODBC data source names.

Example

Supply a connection string in the USING clause of the CREATE SERVER statement to connect to an SAP Sybase IQ database.

```
CREATE SERVER TestSA
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=myhost;Server=TestSA;DBN=sample';
```

Server class ADSODBC

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding Advantage Database Server data types using the following data type conversions.

SAP Sybase IQ data type	ADS default data type
BIT	Logical

Appendix: Accessing Remote Data

SAP Sybase IQ data type	ADS default data type
VARBIT(<i>n</i>)	Binary(<i>n</i>)
LONG VARBIT	Binary(2G)
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Numeric(32)
UNSIGNED TINYINT	Numeric(11)
UNSIGNED SMALLINT	Numeric(11)
UNSIGNED INTEGER	Numeric(11)
UNSIGNED BIGINT	Numeric(32)
CHAR(<i>n</i>)	Character(<i>n</i>)
VARCHAR(<i>n</i>)	VarChar(<i>n</i>)
LONG VARCHAR	VarChar(65000)
NCHAR(<i>n</i>)	NChar(<i>n</i>)
NVARCHAR(<i>n</i>)	NVarChar(<i>n</i>)
LONG NVARCHAR	NVarChar(32500)
BINARY(<i>n</i>)	Binary(<i>n</i>)
VARBINARY(<i>n</i>)	Binary(<i>n</i>)
LONG BINARY	Binary(2G)
DECIMAL(<i>precision</i> , <i>scale</i>)	Numeric(<i>precision</i> +3)
NUMERIC(<i>precision</i> , <i>scale</i>)	Numeric(<i>precision</i> +3)
SMALLMONEY	Money
MONEY	Money
REAL	Double
DOUBLE	Double

SAP Sybase IQ data type	ADS default data type
FLOAT(<i>n</i>)	Double
DATE	Date
TIME	Time
TIMESTAMP	TimeStamp
TIMESTAMP WITH TIMEZONE	Char(254)
BIGDATE	datetime
BIGDATETIME	datetime
XML	Binary(2G)
ST_GEOMETRY	Binary(2G)
UNIQUEIDENTIFIER	Binary(2G)

Server class ASEODBC

A remote server with server class ASEODBC is an Adaptive Server Enterprise (version 10 and later) database server. SAP Sybase IQ requires the installation of the Adaptive Server Enterprise ODBC driver and Open Client connectivity libraries to connect to a remote Adaptive Server Enterprise database server with class ASEODBC.

Notes

- Open Client should be version 11.1.1, EBF 7886 or later. Install Open Client and verify connectivity to the Adaptive Server Enterprise server before you install ODBC and configure SAP Sybase IQ. The Sybase ODBC driver should be version 11.1.1, EBF 7911 or later.
- The local setting of the `quoted_identifier` option controls the use of quoted identifiers for Adaptive Server Enterprise. For example, if you set the `quoted_identifier` option to Off locally, then quoted identifiers are turned off for Adaptive Server Enterprise.
- Configure a user data source in the **Configuration Manager** with the following attributes:
 - **General tab** – Type any value for **Data Source Name**. This value is used in the USING clause of the CREATE SERVER statement.
The server name should match the name of the server in the Sybase interfaces file.
 - **Advanced tab** – Click the **Application Using Threads** and **Enable Quoted Identifiers** options.
 - **Connection tab** – Set the charset field to match your SAP Sybase IQ character set.

Set the language field to your preferred language for error messages.

- **Performance tab** – Set the **Prepare Method** to **2-Full**.

Set the **Fetch Array Size** as large as possible for the best performance. This increases memory requirements since this is the number of rows that must be cached in memory. Adaptive Server Enterprise recommends using a value of 100.

Set **Select Method** to **0-Cursor**.

Set **Packet Size** to as large a value as possible. Adaptive Server Enterprise recommends using a value of -1.

Set **Connection Cache** to 1.

Data type conversions: ODBC and Adaptive Server Enterprise

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding Adaptive Server Enterprise data types. The following table describes the SAP Sybase IQ to Adaptive Server Enterprise data type conversions.

SAP Sybase IQ data type	Adaptive Server Enterprise default data type
BIT	bit
VARBIT(<i>n</i>)	if (n <= 255) varbinary(<i>n</i>) else image
LONG VARBIT	image
TINYINT	tinyint
SMALLINT	smallint
INT, INTEGER	int
BIGINT	numeric(20,0)
UNSIGNED TINYINT	tinyint
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	numeric(11,0)
UNSIGNED BIGINT	numeric(20,0)
CHAR(<i>n</i>)	if (n <= 255) char(<i>n</i>) else text
VARCHAR(<i>n</i>)	if (n <= 255) varchar(<i>n</i>) else text
LONG VARCHAR	text
NCHAR(<i>n</i>)	if (n <= 255) nchar(<i>n</i>) else ntext
NVARCHAR(<i>n</i>)	if (n <= 255) nvarchar(<i>n</i>) else ntext

SAP Sybase IQ data type	Adaptive Server Enterprise default data type
LONG NVARCHAR	ntext
BINARY(<i>n</i>)	if (n <= 255) binary(<i>n</i>) else image
VARBINARY(<i>n</i>)	if (n <= 255) varbinary(<i>n</i>) else image
LONG BINARY	image
DECIMAL(<i>prec, scale</i>)	decimal(<i>prec, scale</i>)
NUMERIC(<i>prec, scale</i>)	numeric(<i>prec, scale</i>)
SMALLMONEY	numeric(10,4)
MONEY	numeric(19,4)
REAL	real
DOUBLE	float
FLOAT(<i>n</i>)	float(<i>n</i>)
DATE	datetime
TIME	datetime
SMALLDATETIME	smalldatetime
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	text
ST_GEOMETRY	image
UNIQUEIDENTIFIER	binary(16)

Example

Supply a connection string in the USING clause of the CREATE SERVER statement to connect to an Adaptive Server Enterprise database.

```
CREATE SERVER TestASE
CLASS 'ASEODBC'
USING 'DRIVER=SYBASE ASE ODBC
Driver;Server=TestASE;Port=5000;Database=testdb;UID=username;PWD=password'
```

Server class DB2ODBC

A remote server with server class DB2ODBC is an IBM DB2 database server.

Notes

- Sybase certifies the use of IBM's DB2 Connect version 5, with fix pack WR09044. Configure and test your ODBC configuration using the instructions for that product. SAP Sybase IQ has no specific requirements for the configuration of IBM DB2 data sources.
- The following is an example of a CREATE EXISTING TABLE statement for an IBM DB2 server with an ODBC data source named mydb2:

```
CREATE EXISTING TABLE ibmcol
AT 'mydb2..sysibm.syscolumns';
```

Data type conversions: IBM DB2

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding IBM DB2 data types. The following table describes the SAP Sybase IQ to IBM DB2 data type conversions.

SAP Sybase IQ data type	IBM DB2 default data type
BIT	smallint
VARBIT(<i>n</i>)	if (n <= 4000) varchar(<i>n</i>) for bit data else long varchar for bit data
LONG VARBIT	long varchar for bit data
TINYINT	smallint
SMALLINT	smallint
INTEGER	int
BIGINT	decimal(20,0)
UNSIGNED TINYINT	int
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	decimal(11,0)
UNSIGNED BIGINT	decimal(20,0)
CHAR(<i>n</i>)	if (n < 255) char(<i>n</i>) else if (n <= 4000) varchar(<i>n</i>) else long varchar
VARCHAR(<i>n</i>)	if (n <= 4000) varchar(<i>n</i>) else long varchar
LONG VARCHAR	long varchar

SAP Sybase IQ data type	IBM DB2 default data type
NCHAR(<i>n</i>)	Not supported
NVARCHAR(<i>n</i>)	Not supported
LONG NVARCHAR	Not supported
BINARY(<i>n</i>)	if (<i>n</i> <= 4000) varchar(<i>n</i>) for bit data else long varchar for bit data
VARBINARY(<i>n</i>)	if (<i>n</i> <= 4000) varchar(<i>n</i>) for bit data else long varchar for bit data
LONG BINARY	long varchar for bit data
DECIMAL(<i>prec, scale</i>)	decimal(<i>prec, scale</i>)
NUMERIC(<i>prec, scale</i>)	decimal(<i>prec, scale</i>)
SMALLMONEY	decimal(10,4)
MONEY	decimal(19,4)
REAL	real
DOUBLE	float
FLOAT(<i>n</i>)	float(<i>n</i>)
DATE	date
TIME	time
TIMESTAMP	timestamp
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	long varchar for bit data
ST_GEOMETRY	long varchar for bit data
UNIQUEIDENTIFIER	varchar(16) for bit data

Server class HANAODBC

A remote server with server class HANAODBC is an SAP HANA database server.

Notes

- The following is an example of a CREATE EXISTING TABLE statement for an SAP HANA database server with an ODBC data source named mySAPHANA:

Appendix: Accessing Remote Data

```
CREATE EXISTING TABLE hanatable
AT 'mySAPHANA..dbo.hanatable';
```

Data type conversions: SAP HANA

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding SAP HANA data types. The following table describes the SAP Sybase IQ to SAP HANA data type conversions.

SAP Sybase IQ data type	SAP HANA default data type
BIT	TINYINT
VARBIT(<i>n</i>)	if (n <= 5000) VARBINARY(<i>n</i>) else BLOB
LONG VARBIT	BLOB
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
UNSIGNED TINYINT	TINYINT
UNSIGNED SMALLINT	INTEGER
UNSIGNED INTEGER	BIGINT
UNSIGNED BIGINT	DECIMAL(20,0)
CHAR(<i>n</i>)	if (n <= 5000) VARCHAR(<i>n</i>) else CLOB
VARCHAR(<i>n</i>)	if (n <= 5000) VARCHAR(<i>n</i>) else CLOB
LONG VARCHAR	CLOB
NCHAR(<i>n</i>)	if (n <= 5000) NVARCHAR(<i>n</i>) else NCLOB
NVARCHAR(<i>n</i>)	if (n <= 5000) NVARCHAR(<i>n</i>) else NCLOB
LONG NVARCHAR	NCLOB
BINARY(<i>n</i>)	if (n <= 5000) VARBINARY(<i>n</i>) else BLOB
VARBINARY(<i>n</i>)	if (n <= 5000) VARBINARY(<i>n</i>) else BLOB
LONG BINARY	BLOB
DECIMAL(<i>precision</i> , <i>scale</i>)	DECIMAL(<i>precision</i> , <i>scale</i>)
NUMERIC(<i>precision</i> , <i>scale</i>)	DECIMAL(<i>precision</i> , <i>scale</i>)

SAP Sybase IQ data type	SAP HANA default data type
SMALLMONEY	DECIMAL(13,4)
MONEY	DECIMAL(19,4)
REAL	REAL
DOUBLE	FLOAT
FLOAT(<i>n</i>)	FLOAT
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
TIMESTAMP WITH TIMEZONE	VARCHAR(254)
XML	BLOB
ST_GEOMETRY	BLOB
UNIQUEIDENTIFIER	VARBINARY(16)

Server class IQODBC

A remote server with server class IQODBC is an SAP Sybase IQ database server. No special requirements exist for the configuration of an SAP Sybase IQ data source.

To access SAP Sybase IQ database servers that support multiple databases, create an ODBC data source name defining a connection to each database. Execute a CREATE SERVER statement for each of these ODBC data source names.

Server class MSACCESSODBC

Access databases are stored in a .mdb file. Using the ODBC manager, create an ODBC data source and map it to one of these files. A new .mdb file can be created through the ODBC manager. This database file becomes the default if you don't specify a different default when you create a table through SAP Sybase IQ.

Assuming an ODBC data source named access, you can use any of the following statements to access data:

- CREATE TABLE tabl (a int, b char(10))
AT 'access...tabl';
- CREATE TABLE tabl (a int, b char(10))
AT 'access;d:\\pcdb\\data.mdb;;tabl';
- CREATE EXISTING TABLE tabl
AT 'access;d:\\pcdb\\data.mdb;;tabl';

Access does not support the owner name qualification; leave it empty.

Data type conversions: Microsoft Access

SAP Sybase IQ data type	Microsoft Access default data type
BIT	TINYINT
VARBIT(<i>n</i>)	if (n <= 4000) BINARY(<i>n</i>) else IMAGE
LONG VARBIT	IMAGE
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	DECIMAL(19,0)
UNSIGNED TINYINT	TINYINT
UNSIGNED SMALLINT	INTEGER
UNSIGNED INTEGER	DECIMAL(11,0)
UNSIGNED BIGINT	DECIMAL(20,0)
CHAR(<i>n</i>)	if (n < 255) CHARACTER(<i>n</i>) else TEXT
VARCHAR(<i>n</i>)	if (n < 255) CHARACTER(<i>n</i>) else TEXT
LONG VARCHAR	TEXT
NCHAR(<i>n</i>)	Not supported
NVARCHAR(<i>n</i>)	Not supported
LONG NVARCHAR	Not supported
BINARY(<i>n</i>)	if (n <= 4000) BINARY(<i>n</i>) else IMAGE
VARBINARY(<i>n</i>)	if (n <= 4000) BINARY(<i>n</i>) else IMAGE
LONG BINARY	IMAGE
DECIMAL(<i>precision, scale</i>)	DECIMAL(<i>precision, scale</i>)
NUMERIC(<i>precision, scale</i>)	DECIMAL(<i>precision, scale</i>)
SMALLMONEY	MONEY
MONEY	MONEY
REAL	REAL

SAP Sybase IQ data type	Microsoft Access default data type
DOUBLE	FLOAT
FLOAT(<i>n</i>)	FLOAT
DATE	DATETIME
TIME	DATETIME
TIMESTAMP	DATETIME
TIMESTAMP WITH TIMEZONE	CHARACTER(254)
XML	XML
ST_GEOMETRY	IMAGE
UNIQUEIDENTIFIER	BINARY(16)

Server class MSSODBC

The server class MSSODBC is used to access Microsoft SQL Server through one of its ODBC drivers.

Notes

- Versions of Microsoft SQL Server ODBC drivers that have been used are:
 - Microsoft SQL Server ODBC Driver Version 06.01.7601
 - Microsoft SQL Server Native Client Version 10.00.1600
- The following is an example for Microsoft SQL Server:

```
CREATE SERVER mysqlserver
CLASS 'MSSODBC'
USING 'DSN=MSSODBC_cli';

CREATE EXISTING TABLE accounts
AT 'mysqlserver.master.dbo.accounts';
```

- The local setting of the `quoted_identifier` option controls the use of quoted identifiers for Microsoft SQL Server. For example, if you set the `quoted_identifier` option to Off locally, then quoted identifiers are turned off for Microsoft SQL Server.

Data type conversions: Microsoft SQL Server

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding Microsoft SQL Server data types using the following data type conversions.

SAP Sybase IQ data type	Microsoft SQL Server default data type
BIT	bit

SAP Sybase IQ data type	Microsoft SQL Server default data type
VARBIT(<i>n</i>)	if (n <= 255) varbinary(<i>n</i>) else image
LONG VARBIT	image
TINYINT	tinyint
SMALLINT	smallint
INTEGER	int
BIGINT	numeric(20,0)
UNSIGNED TINYINT	tinyint
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	numeric(11,0)
UNSIGNED BIGINT	numeric(20,0)
CHAR(<i>n</i>)	if (n <= 255) char(<i>n</i>) else text
VARCHAR(<i>n</i>)	if (n <= 255) varchar(<i>n</i>) else text
LONG VARCHAR	text
NCHAR(<i>n</i>)	if (n <= 4000) nchar(<i>n</i>) else ntext
NVARCHAR(<i>n</i>)	if (n <= 4000) nvarchar(<i>n</i>) else ntext
LONG NVARCHAR	ntext
BINARY(<i>n</i>)	if (n <= 255) binary(<i>n</i>) else image
VARBINARY(<i>n</i>)	if (n <= 255) varbinary(<i>n</i>) else image
LONG BINARY	image
DECIMAL(<i>precision, scale</i>)	decimal(<i>precision, scale</i>)
NUMERIC(<i>precision, scale</i>)	numeric(<i>precision, scale</i>)
SMALLMONEY	smallmoney
MONEY	money
REAL	real
DOUBLE	float
FLOAT(<i>n</i>)	float(<i>n</i>)

SAP Sybase IQ data type	Microsoft SQL Server default data type
DATE	datetime
TIME	datetime
SMALLDATETIME	smalldatetime
DATETIME	datetime
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	xml
ST_GEOMETRY	image
UNIQUEIDENTIFIER	binary(16)

Server class MYSQLODBC

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding MySQL data types using the following data type conversions.

SAP Sybase IQ data type	MySQL default data type
BIT	bit(1)
VARBIT(<i>n</i>)	if (<i>n</i> <= 4000) varbinary(<i>n</i>) else longblob
LONG VARBIT	longblob
TINYINT	tinyint unsigned
SMALLINT	smallint
INTEGER	int
BIGINT	bigint
UNSIGNED TINYINT	tinyint unsigned
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	bigint
UNSIGNED BIGINT	decimal(20,0)

SAP Sybase IQ data type	MySQL default data type
CHAR(<i>n</i>)	if (<i>n</i> < 255) char(<i>n</i>) else if (<i>n</i> <= 4000) var-char(<i>n</i>) else longtext
VARCHAR(<i>n</i>)	if (<i>n</i> <= 4000) var-char(<i>n</i>) else longtext
LONG VARCHAR	longtext
NCHAR(<i>n</i>)	if (<i>n</i> < 255) national character(<i>n</i>) else if (<i>n</i> <= 4000) national character varying(<i>n</i>) else longtext
NVARCHAR(<i>n</i>)	if (<i>n</i> <= 4000) national character varying(<i>n</i>) else longtext
LONG NVARCHAR	longtext
BINARY(<i>n</i>)	if (<i>n</i> <= 4000) varbinary(<i>n</i>) else longblob
VARBINARY(<i>n</i>)	if (<i>n</i> <= 4000) varbinary(<i>n</i>) else longblob
LONG BINARY	longblob
DECIMAL(<i>precision</i> , <i>scale</i>)	decimal(<i>precision</i> , <i>scale</i>)
NUMERIC(<i>precision</i> , <i>scale</i>)	decimal(<i>precision</i> , <i>scale</i>)
SMALLMONEY	decimal(10,4)
MONEY	decimal(19,4)
REAL	real
DOUBLE	float
FLOAT(<i>n</i>)	float(<i>n</i>)
DATE	date
TIME	time
TIMESTAMP	datetime

SAP Sybase IQ data type	MySQL default data type
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	longblob
ST_GEOMETRY	longblob
UNIQUEIDENTIFIER	varbinary(16)

Example

Supply a connection string in the USING clause of the CREATE SERVER statement to connect to a MySQL database.

```
CREATE SERVER TestMySQL
CLASS 'MSDQL'
USING 'DRIVER=MySQL ODBC 5.1
Driver;DATABASE=mydatabase;SERVER=mysqlHost;UID=me;PWD=secret'
```

Server class ODBC

ODBC data sources that do not have their own server class use server class ODBC. You can use any ODBC driver. Sybase certifies the following ODBC data sources:

- Microsoft Excel (Microsoft 3.51.171300)
- Microsoft FoxPro (Microsoft 3.51.171300)
- Lotus Notes SQL

The latest versions of Microsoft ODBC drivers can be obtained through the Microsoft Data Access Components (MDAC) distribution found at the Microsoft Download Center. The Microsoft driver versions listed above are part of MDAC 2.0.

Microsoft Excel (Microsoft 3.51.171300)

With Excel, each Excel workbook is logically considered to be a database holding several tables. Tables are mapped to sheets in a workbook. When you configure an ODBC data source name in the ODBC driver manager, you specify a default workbook name associated with that data source. However, when you execute a CREATE TABLE statement, you can override the default and specify a workbook name in the location string. This allows you to use a single ODBC DSN to access all of your Excel workbooks.

Create a remote server named excel that connects to the Microsoft Excel ODBC driver.

```
CREATE SERVER excel
CLASS 'ODBC'
USING 'DRIVER=Microsoft Excel Driver (*.xls);DBQ=d:\
\work1.xls;READONLY=0;DriverID=790'
```

To create a workbook named `work1.xls` with a sheet (table) called `mywork`:

Appendix: Accessing Remote Data

```
CREATE TABLE mywork (a int, b char(20))
AT 'excel;d:\\work1.xls;;mywork';
```

To create a second sheet (or table) execute a statement such as:

```
CREATE TABLE mywork2 (x float, y int)
AT 'excel;d:\\work1.xls;;mywork2';
```

You can import existing sheets into SAP Sybase IQ using CREATE EXISTING, under the assumption that the first row of your sheet contains column names.

```
CREATE EXISTING TABLE mywork
AT 'excel;d:\\work1;;mywork';
```

If SAP Sybase IQ reports that the table is not found, you may need to explicitly state the column and row range you want to map to. For example:

```
CREATE EXISTING TABLE mywork
AT 'excel;d:\\work1;;mywork$';
```

Adding the \$ to the sheet name indicates that the entire worksheet should be selected.

Note in the location string specified by AT that a semicolon is used instead of a period for field separators. This is because periods occur in the file names. Excel does not support the owner name field so leave this blank.

Deletes are not supported. Also some updates may not be possible since the Excel driver does not support positioned updates.

Example

The following statements create a database server called TestExcel that uses an ODBC DSN to access the Excel workbook LogFile.xlsx and import its sheet it into SAP Sybase IQ.

```
CREATE SERVER TestExcel
CLASS 'ODBC'
USING 'DRIVER=Microsoft Excel Driver (*.xls);DBQ=c:\\temp\\
\LogFile.xlsx;READONLY=0;DriverID=790'

CREATE EXISTING TABLE MyWorkbook
AT 'TestExcel;c:\\temp\\LogFile.xlsx;;Logfile$';

SELECT * FROM MyWorkbook;
```

Microsoft FoxPro (Microsoft 3.51.171300)

You can store FoxPro tables together inside a single FoxPro database file (.dbc), or, you can store each table in its own separate .dbf file. When using .dbf files, be sure the file name is filled into the location string; otherwise the directory that SAP Sybase IQ was started in is used.

```
CREATE TABLE fox1 (a int, b char(20))
AT 'foxpro;d:\\pcdb;;fox1';
```

This statement creates a file named d:\pcdb\fox1.dbf when you choose the **Free Table Directory** option in the ODBC Driver Manager.

Lotus Notes SQL

To obtain this driver, go to the Lotus NotesSQL web site at <http://www.ibm.com/developerworks/lotus/products/notesdomino/notesql/>. Read the documentation that is included with it for an explanation of how Notes data maps to relational tables. You can easily map SAP Sybase IQ tables to Notes forms.

Here is how to set up SAP Sybase IQ to access your Lotus Notes contacts.

- Make sure that the Lotus Notes program folder is in your path (for example, C:\Program Files (x86)\IBM\Lotus\Notes).
- Create a 32-bit ODBC data source using the NotesSQL ODBC driver. Use the `names.nsf` database for this example. The **Map Special Characters** option should be turned on. For this example, the **Data Source Name** is `my_notes_dsn`.
- Create a remote data access server using Interactive SQL connected to a 32-bit database server. Here is an example:

```
CREATE SERVER NotesContacts
CLASS 'ODBC'
USING 'my_notes_dsn';
```

- Create an external login for the Lotus Notes server. Here is an example:

```
CREATE EXTERNLOGIN "DBA" TO "NotesContacts"
REMOTE LOGIN 'John Doe/SYBASE' IDENTIFIED BY 'MyNotesPassword';
```

- Map some columns of the Person form into an SAP Sybase IQ table:

```
CREATE EXISTING TABLE PersonDetails
( DisplayName CHAR(254),
  DisplayMailAddress CHAR(254),
  JobTitle CHAR(254),
  CompanyName CHAR(254),
  Department CHAR(254),
  Location CHAR(254),
  OfficePhoneNumber CHAR(254) )
AT 'NotesContacts...Person';
```

- Query the table:

```
SELECT * FROM PersonDetails
WHERE Location LIKE 'Waterloo%';
```

Server class ORAODBC

A remote server with server class ORAODBC is an Oracle Database version 8.0 or later.

Notes

- Sybase certifies the use of the Oracle Database version 8.0.03 ODBC driver. Configure and test your ODBC configuration using the instructions for that product.
- The following is an example of a CREATE EXISTING TABLE statement for an Oracle Database server named myora:

Appendix: Accessing Remote Data

```
CREATE EXISTING TABLE employees
AT 'myora.database.owner.employees';
```

Data type conversions: Oracle Database

When you execute a CREATE TABLE statement, SAP Sybase IQ automatically converts the data types to the corresponding Oracle Database data types using the following data type conversions.

SAP Sybase IQ data type	Oracle Database data type
BIT	number(1,0)
VARBIT(<i>n</i>)	if (n <= 255) raw(<i>n</i>) else long raw
LONG VARBIT	long raw
TINYINT	number(3,0)
SMALLINT	number(5,0)
INTEGER	number(11,0)
BIGINT	number(20,0)
UNSIGNED TINYINT	number(3,0)
UNSIGNED SMALLINT	number(5,0)
UNSIGNED INTEGER	number(11,0)
UNSIGNED BIGINT	number(20,0)
CHAR(<i>n</i>)	if (n <= 255) char(<i>n</i>) else long
VARCHAR(<i>n</i>)	if (n <= 2000) varchar(<i>n</i>) else long
LONG VARCHAR	long
NCHAR(<i>n</i>)	if (n <= 255) nchar(<i>n</i>) else nclob
NVARCHAR(<i>n</i>)	if (n <= 2000) nvarchar(<i>n</i>) else nclob
LONG NVARCHAR	nclob

SAP Sybase IQ data type	Oracle Database data type
BINARY(<i>n</i>)	if (<i>n</i> > 255) long raw else raw(<i>n</i>)
VARBINARY(<i>n</i>)	if (<i>n</i> > 255) long raw else raw(<i>n</i>)
LONG BINARY	long raw
DECIMAL(<i>precision</i> , <i>scale</i>)	number(<i>precision</i> , <i>scale</i>)
NUMERIC(<i>precision</i> , <i>scale</i>)	number(<i>precision</i> , <i>scale</i>)
SMALLMONEY	numeric(13,4)
MONEY	number(19,4)
REAL	real
DOUBLE	float
FLOAT(<i>n</i>)	float
DATE	date
TIME	date
TIMESTAMP	date
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	long raw
ST_GEOMETRY	long raw
UNIQUEIDENTIFI- ER	raw(16)

Example

Supply a connection string in the USING clause of the CREATE SERVER statement to connect to an Oracle database.

```
CREATE SERVER TestOracle
CLASS 'ORAODBC'
USING 'DRIVER=Oracle ODBC
Driver;DBQ=mydatabase;UID=username;PWD=password'
```

Remote Servers

Before remote objects can be mapped to a local proxy table, define the remote server where the remote object is located.

Create Remote Servers

Use the **CREATE SERVER** statement to set up remote server definitions.

For some systems, including SAP Sybase IQ and SQL Anywhere, each data source describes a database, so a separate remote server definition is needed for each database.

See also

- *CREATE SERVER Statement* on page 966

Before You Access Remote Oracle Data

To access remote Oracle data, configure your system with the prerequisite software.

1. *Check for Prerequisites*

Check your system for the software components required to access Oracle data using Component Integration Services (CIS).

2. *Create an Oracle Data Source Name*

Use the `iqdsn` utility to create an entry in the `.odbc.ini` file.

3. *Set Environment Variables for Oracle Data Access*

Before starting the SAP Sybase IQ server to access Oracle data, you must set certain environment variables.

4. *Start the SAP Sybase IQ Server*

Start the SAP Sybase IQ server that you will use as a front end to access Oracle data.

Check for Prerequisites

Check your system for the software components required to access Oracle data using Component Integration Services (CIS).

Prerequisites are:

- An Oracle database
- Oracle client software (basic package), including a `network/admin/tnsnames.ora` file.
- A platform-specific driver (installed with SAP Sybase IQ):

Platform	File
AIX 64	<code>\$IQDIR16/libxx/libdbor-aodbc12_r.so</code>

Platform	File
HPiUX	\$IQDIR16/libxx/libdbor-aodbc12_r.so.1
Linux64	\$IQDIR16/libxx/libdbor-aodbc12_r.so.1
SunOS64	\$IQDIR16/libxx/libdbor-aodbc12_r.so.1
WinAMD64	%\$IQDIR16%\bin64\dbor-aodbc12.dll

Create an Oracle Data Source Name

Use the `iqdsn` utility to create an entry in the `.odbc.ini` file.

1. Display Oracle connection keywords:

```
% iqdsn -cl -or
```

```
Driver
UserID          UID
Password        PWD
SID             SID
Encrypted Password ENP
ProcResults     PROC
ArraySize       SIZE
EnableMSDTC     EDTC
ProcOwner       POWNER
```

2. Create an `.odbc.ini` file entry:

```
% iqdsn -or -y -w "MyOra2" -c
"UID=system;PWD=manager;SID=QAORA"
```

```
[MyOra2]
Driver=/Sybase/IQ-16_0/lib64/libdboraodbc12_r.so
UserID=system
Password=manager
SID=QAORA
```

Set Environment Variables for Oracle Data Access

Before starting the SAP Sybase IQ server to access Oracle data, you must set certain environment variables.

Set these variables for Oracle access:

- ORACLE_HOME
setenv ORACLE_HOME
- ODBCINI

Appendix: Accessing Remote Data

```
setenv ODBCINI <location of .odbc.ini file with Oracle entry>
```

- The library path variable for your platform

Platform	Command
AIX	<pre>setenv LIBPATH <path to platform-specific Oracle client directory> \$LIBPATH</pre>
Other UNIX platforms	<pre>setenv LD_LIBRARY_PATH <path to platform-specific Oracle client directory>;\$LD_LIBRARY_PATH</pre>

Start the SAP Sybase IQ Server

Start the SAP Sybase IQ server that you will use as a front end to access Oracle data.

```
start_iq -n myserver
```

Connecting to an Oracle Database

Connect SAP Sybase IQ to remote Oracle data via Component Integration Services.

Prerequisites

Log in to **dbisql** or **iqisql**.

Task

1. Create a server using the data source name from the `.odbc.ini` file:

```
CREATE SERVER myora CLASS 'oraodbc' USING 'MyOra2'
```

2. Create an external login:

```
CREATE EXTERNLOGIN DBA TO myora REMOTE LOGIN system IDENTIFIED BY manager
```

3. Confirm the connection:

```
sp_remote_tables myora
```

4. Create a table of Oracle data:

```
CREATE EXISTING TABLE my_oratable at 'myora..system.oratable'
```

5. Verify that the connection works by selecting data:

```
SELECT * FROM my_oratable
```


Troubleshoot Oracle Database Access

If Oracle data access returns errors, check the appropriate configuration component.

1. *Error Loading Driver*

A driver load error may indicate a problem with an environment variable or the configuration information file.

2. *Error Resolving Connect Identifier*

An error resolving the connect identifier may be a problem with the Oracle definition, an environment variable, or the configuration information file.

Error Loading Driver

A driver load error may indicate a problem with an environment variable or the configuration information file.

If you receive a `Can't load driver` error:

- Check that the `.odbc.ini` entry lists the correct driver.
- Check that the Oracle client software is added to the `LD_LIBRARY_PATH` definition.

Error Resolving Connect Identifier

An error resolving the connect identifier may be a problem with the Oracle definition, an environment variable, or the configuration information file.

If you see the error `ORA-12154: TNS:could not resolve the connect identifier`:

- Check that the Oracle definition is correct.
- Check that `ORACLE_HOME` is set correctly.
- Check that the gateway system identifier (SID) entered in `.odbc.ini` is correct.

Loading Remote Data Without Native Classes

Load data by using DirectConnect™.

Native classes use DirectConnect to access remote data sources:

- On 64-bit UNIX platforms
- On 32-bit platforms where no ODBC driver is available (for example, Microsoft SQL Server)

Loading MS SQL Server Data into an SAP Sybase IQ Server on UNIX

This remote data example loads MS SQL Server data into an SAP Sybase IQ server on UNIX.

For this example, assume that:

Appendix: Accessing Remote Data

- An Enterprise Connect Data Access (ECDA) server named *mssql* exists on UNIX host *myhostname*, port 12530.
- The data is to be retrieved from an MS SQL server named *2000* on host *myhostname*, port 1433.

1. Using DirectConnect documentation, configure DirectConnect for your data source.
2. Make sure that ECDA server (*mssql*) is listed in the SAP Sybase IQ interfaces file:

```
mssql
master tcp ether myhostname 12530
query tcp ether myhostname 12530
```

3. Add a new user, using the user ID and password for server *mssql*:

```
isql -Udba -Psql -Stst_iqdemo
grant connect to chill identified by chill
grant dba to chill
```

4. Log in as the new user to create a local table on SAP Sybase IQ:

```
isql -Uchill -Pchill -Stst_iqdemo
create table billing(status char(1), name varchar(20), telno int)
```

5. Insert data:

```
insert into billing location 'mssql.pubs' { select * from
billing }
```

Querying Data Without Native Classes

Follow these guidelines to query data without native classes.

1. Configure ASE/CIS with a remote server and proxy to connect via DirectConnect. For example, use DirectConnect for Oracle to the Oracle server.
2. Configure SAP Sybase IQ with a remote server using the ASEJDBC class to the Adaptive Server server. (The ASEODBC class is unavailable because there is no 64-bit Unix ODBC driver for Adaptive Server.)
3. Use the **CREATE EXISTING TABLE** statement to create proxy tables pointing to the proxy tables in ASE which in turn point to Oracle.

Querying Remote Data Using DirectConnect and Proxy Table from UNIX

Query data using DirectConnect.

This example shows how to access MS SQL Server data. For this example, assume the following:

- An SAP Sybase IQ server on host *myhostname*, port 7594.
- An Adaptive Server server on host *myhostname*, port 4101.
- An Enterprise Connect Data Access (ECDA) server exists named *mssql* on host *myhostname*, port 12530.
- The data is to be retrieved from an MS SQL server named *2000* on host *myhostname*, port 1433.

Setting Up Adaptive Server to Query MS SQL Server

Set up Adaptive Server and Component Integration Services (CIS) to query MS SQL Server through DirectConnect.

For this example, assume that the server name is *jones_1207*.

1. Add an entry to the Adaptive Server interfaces file to connect to *mssql*:

```
mssql
master tcp ether hostname 12530
query tcp ether hostname 12530
```

2. Enable CIS and remote procedure call handling from the ASE server. For example, if CIS is already enabled as the default:

```
sp_configure 'enable cis'
Parameter Name Default Memory Used Config Value Run Value
enable cis          1          1          1          0
1
(1 row affected)
(return status=0)

sp_configure 'cis rpc handling', 1
Parameter Name Default Memory Used Config Value Run Value
enable cis          0          1          0          0
0
(1 row affected)
Configuration option changed. The SQL Server need not be restarted
since the option is dynamic.
```

3. Add the DirectConnect server to the Adaptive Server server's SYSSERVERS system table.

```
sp_addserver mssql, direct_connect, mssql
Adding server 'mssql', physical name 'mssql'
Server added.
(Return status=0)
```

4. Create the user in Adaptive Server that will be used in SAP Sybase IQ to connect to Adaptive Server.

```
sp_addlogin tst, tsttst
Password correctly set.
Account unlocked. New login created.
(return status = 0)

grant role sa_role to tst
use tst_db
sp_adduser tst
```

Appendix: Accessing Remote Data

```
New user added.  
(return status = 0)
```

5. Add an external login from the master database:

```
use master  
sp_addexternlogin mssql, tst, chill, chill
```

```
User 'tst' will be known as 'chill' in remote server 'mssql'.  
(return status = 0)
```

6. Create an ASE proxy table as the added user from the desired database:

```
isql -Utst -Ttsttst  
use test_db  
create proxy_table billing_tst at 'mssql.pubs..billing'  
select * from billing_tst
```

status	name	telno
-----	-----	-----
D	BOTANICALLY	1
B	BOTANICALL	2

(2 rows affected)

Setting up SAP Sybase IQ to Connect to the Adaptive Server Server

Follow these steps to query Adaptive Server data.

1. Add an entry to the SAP Sybase IQ interfaces file:

```
jones_1207  
master tcp ether jones 4101  
query tcp ether jones 4101
```

2. Create the user to connect to Adaptive Server:

```
GRANT CONNECT TO tst IDENTIFIED BY tsttst  
GRANT dba TO tst
```

3. Log in as the added user to create the 'asejdbc' server class and add external login:

```
isql -Utst -Ptsttst -Stst_iqdemo  
CREATE SERVER jones_1207 CLASS 'asejdbc' USING 'jones:4101/tst_db'  
CREATE EXISTING TABLE billing_iq AT  
'jones_1207.tst_db..billing_txt'  
SELECT * from billing_iq
```

status	name	telno
-----	-----	-----
D	BOTANICALLY	1
B	BOTANICALL	2

(2 rows affected)

Delete Remote Servers

Use the **DROP SERVER** statement to delete a remote server from the ISYSSERVER system table.

All remote tables defined on that server must already be dropped for this action to succeed.

Example

This statement drops the server named RemoteSA:

```
DROP SERVER RemoteSA;
```

See also

- *DROP SERVER Statement* on page 985

Alter Remote Servers

Use the **ALTER SERVER** statement to modify the attributes of a server. These changes do not take effect until the next connection to the remote server.

Execute an **ALTER SERVER** statement.

The following statement changes the server class of the server named RemoteASE to aseodbc. In this example, the Data Source Name for the server is RemoteASE

```
ALTER SERVER RemoteASE  
CLASS 'aseodbc';
```

See also

- *ALTER SERVER Statement* on page 961

Listing the tables on a remote server (SQL)

You can view a limited or comprehensive list of all the tables on a remote server using a system procedure.

Prerequisites

None.

Task

Call the `sp_remote_tables` system procedure to return a list of the tables on a remote server.

If you specify *@table_name* or *@table_owner*, the list of tables is limited to only those that match.

A list of all the tables, or a limited list of tables, is returned.

Remote server capabilities

The `sp_servercaps` system procedure displays information about a remote server's capabilities. SAP Sybase IQ uses this capability information to determine how much of a SQL statement can be passed to a remote server.

You can also view capability information for remote servers by querying the `SYSCAPABILITY` and `SYSCAPABILITYNAME` system views. These system views are empty until after SAP Sybase IQ first connects to a remote server.

Appendix: Accessing Remote Data

When using the `sp_servercaps` system procedure, the *server-name* specified must be the same *server-name* used in the `CREATE SERVER` statement.

Execute the stored procedure `sp_servercaps` as follows:

```
CALL sp_servercaps('server-name');
```

External Logins

SAP Sybase IQ uses the names and passwords of its clients when it connects to a remote server on behalf of those clients. However, this behavior can be overridden by creating external logins.

External logins are alternate login names and passwords that are used when communicating with a remote server.

When SAP Sybase IQ connects to the remote server, **INSERT...LOCATION** uses the remote login for the user ID of the current connection, if a remote login has been created with **CREATE EXTERNLOGIN** and the remote server has been defined with a **CREATE SERVER** statement.

If the remote server is not defined, or a remote login has not been created for the user ID of the current connection, SAP Sybase IQ connects using the user ID and password of the current connection.

Note: If you rely on the default user ID and password, and a user changes the password, you must stop and restart the server before the new password takes effect on the remote server. Remote logins created with **CREATE EXTERNLOGIN** are unaffected by changes to the password for the default user ID.

If you use an integrated login, the SAP Sybase IQ name and password of the SAP Sybase IQ client is the same as the database login ID and password that the SAP Sybase IQ user ID maps to in `syslogins`.

Proxy tables

Location transparency of remote data is enabled by creating a local **proxy table** that maps to the remote object. You can use a proxy table to access any object (including tables, views, and materialized views) that the remote database exports as a candidate for a proxy table. Use one of the following statements to create a proxy table:

- If the table already exists at the remote storage location, use the `CREATE EXISTING TABLE` statement. This statement defines the proxy table for an existing table on the remote server.
- If the table does not exist at the remote storage location, use the `CREATE TABLE` statement. This statement creates a new table on the remote server, and also defines the proxy table for that table.

Note: You cannot modify data in a proxy table when you are within a savepoint.

When a trigger is fired on a proxy table, the permissions used are those of the user who caused the trigger to fire, not those of the proxy table owner.

Proxy table locations

The AT keyword is used with both the CREATE TABLE and the CREATE EXISTING TABLE statements to define the location of an existing object. This location string has four components, each separated by either a period or a semicolon. The semicolon delimiter allows file names and extensions to be used in the database and owner fields.

The syntax of the AT clause is:

```
... AT 'server.database.owner.table-name'
```

- **server** – This is the name by which the server is known in the current database, as specified in the CREATE SERVER statement. This field is mandatory for all remote data sources.
- **database** – The meaning of the database field depends on the data source. Sometimes this field does not apply and should be left empty. The delimiter is still required, however.

If the data source is Adaptive Server Enterprise, *database* specifies the database where the table exists. For example master or pubs2.

If the data source is SAP Sybase IQ, this field does not apply; leave it empty.

If the data source is Excel, Lotus Notes, or Access, you must include the name of the file containing the table. If the file name includes a period, use the semicolon delimiter.

- **owner** – If the database supports the concept of ownership, this field represents the owner name. This field is only required when several owners have tables with the same name.
- **table-name** – This field specifies the name of the table. For an Excel spreadsheet, this is the name of the sheet in the workbook. If *table-name* is left empty, the remote table name is assumed to be the same as the local proxy table name.

Examples

The following examples illustrate the use of location strings:

- SAP Sybase IQ:

```
'RemoteSA..GROUPO.Employees'
```

- Adaptive Server Enterprise:

```
'RemoteASE.pubs2.dbo.publishers'
```

- Excel:

```
'RemoteExcel;d:\pcdb\quarter3.xls;;sheet1$'
```

- Access:

```
'RemoteAccessDB;\\server1\production\inventory.mdb;;parts'
```

Creating proxy tables (SQL)

You can create proxy tables in Interactive SQL using either the CREATE TABLE or CREATE EXISTING TABLE statement.

Prerequisites

You must have the CREATE PROXY TABLE system privilege to create proxy tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

Task

The CREATE TABLE statement creates a new table on the remote server, and defines the proxy table for that table when you use the AT clause. Columns are defined using SAP Sybase IQ data types. SAP Sybase IQ automatically converts the data into the remote server's native types.

If you use the CREATE TABLE statement to create both a local and remote table, and then subsequently use the DROP TABLE statement to drop the proxy table, the remote table is also dropped. You can, however, use the DROP TABLE statement to drop a proxy table created using the CREATE EXISTING TABLE statement. In this case, the remote table is not dropped.

The CREATE EXISTING TABLE statement creates a proxy table that maps to an existing table on the remote server. SAP Sybase IQ derives the column attributes and index information from the object at the remote location.

1. Connect to the host database.
2. Execute a CREATE EXISTING TABLE statement.

The proxy table is created.

Note: Before using a new proxy table on a multiplex secondary server, disconnect and reconnect to the server.

See also

- *CREATE EXISTING TABLE Statement* on page 964
- *CREATE TABLE Statement* on page 968

List the columns on a remote table

Before you execute a CREATE EXISTING TABLE statement, it may be helpful to get a list of the columns that are available on a remote table. The sp_remote_columns system procedure

produces a list of the columns on a remote table and a description of those data types. The following is the syntax for the `sp_remote_columns` system procedure:

```
CALL sp_remote_columns( @server_name, @table_name [, @table_owner [,
@table_qualifier ] ] )
```

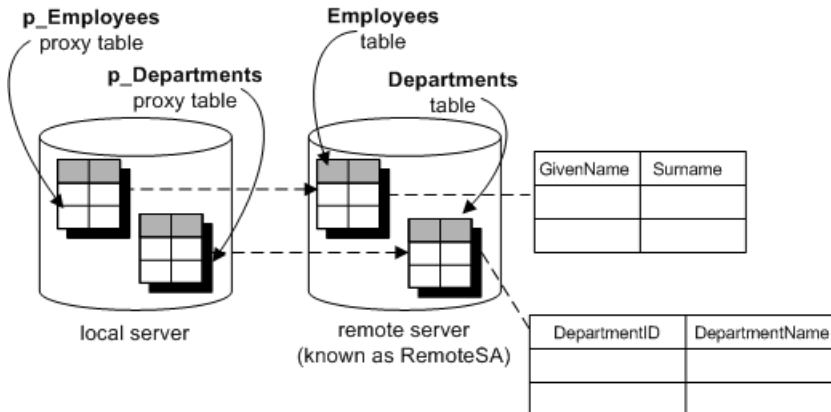
If a table name, owner, or database name is given, the list of columns is limited to only those that match.

For example, the following returns a list of the columns in the `sysobjects` table in the production database on an Adaptive Server Enterprise server named `asetest`:

```
CALL sp_remote_columns('asetest', 'sysobjects', null, 'production');
```

Joins between remote tables

The following figure illustrates proxy tables on a local database server that are mapped to the remote tables `Employees` and `Departments` of the SAP Sybase IQ sample database on the remote server `RemoteSA`.



You can use joins between tables on different SAP Sybase IQ databases. The following example is a simple case using just one database to illustrate the principles.

Example

Perform a join between two remote tables:

1. Create a new database named `empty.db`.

This database holds no data. It is used only to define the remote objects, and to access the SAP Sybase IQ sample database.

2. Start a database server running the `empty.db`. You can do this by running the following command:

```
iqsrv16 empty
```

3. From Interactive SQL, connect to `empty.db` as user `DBA`.
4. In the new database, create a remote server named `RemoteSA`. Its server class is `SAODBC`, and the connection string refers to the SAP Sybase IQ 16 Demo ODBC data source:

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'SAP Sybase IQ 16 Demo';
```

5. In this example, you use the same user ID and password on the remote database as on the local database, so no external logins are needed.

Sometimes you must provide a user ID and password when connecting to the database at the remote server. In the new database, you could create an external login to the remote server. For simplicity in this example, the local login name and the remote user ID are both DBA:

```
CREATE EXTERNLOGIN DBA
TO RemoteSA
REMOTE LOGIN DBA
IDENTIFIED BY sql;
```

6. Define the p_Employees proxy table:

```
CREATE EXISTING TABLE p_Employees
AT 'RemoteSA..GROUPO.Employees';
```

7. Define the p_Departments proxy table:

```
CREATE EXISTING TABLE p_Departments
AT 'RemoteSA..GROUPO.Departments';
```

8. Use the proxy tables in the SELECT statement to perform the join.

```
SELECT GivenName, Surname, DepartmentName
FROM p_Employees JOIN p_Departments
ON p_Employees.DepartmentID = p_Departments.DepartmentID
ORDER BY Surname;
```

Joins between tables from multiple local databases

An SAP Sybase IQ server may have several local databases running at one time. By defining tables in other local SAP Sybase IQ databases as remote tables, you can perform cross-database joins.

Example

Suppose you are using database db1, and you want to access data in tables in database db2. You need to set up proxy table definitions that point to the tables in database db2. For example, on an SAP Sybase IQ server named RemoteSA, you might have three databases available: db1, db2, and db3.

1. If you are using ODBC, create an ODBC data source name for each database you will be accessing.
2. Connect to the database from which you will be performing the join. For example, connect to db1.
3. Perform a CREATE SERVER statement for each other local database you will be accessing. This sets up a **loopback** connection to your SAP Sybase IQ server.

```
CREATE SERVER remote_db2
CLASS 'SAODBC'
USING 'RemoteSA_db2';
CREATE SERVER remote_db3
```

```
CLASS 'SAODBC'
USING 'RemoteSA_db3';
```

4. Create proxy table definitions by executing CREATE EXISTING TABLE statements for the tables in the other databases you want to access.

```
CREATE EXISTING TABLE Employees
AT 'remote_db2...Employees';
```

Native statements and remote servers

Use the FORWARD TO statement to send one or more statements to the remote server in its native syntax. This statement can be used in two ways:

- To send a statement to a remote server.
- To place SAP Sybase IQ into passthrough mode for sending a series of statements to a remote server.

The FORWARD TO statement can be used to verify that a server is configured correctly. If you send a statement to the remote server and SAP Sybase IQ does not return an error message, the remote server is configured correctly.

The FORWARD TO statement cannot be used within procedures or batches.

If a connection cannot be made to the specified server, a message is returned to the user. If a connection is made, any results are converted into a form that can be recognized by the client program.

Example 1

The following statement verifies connectivity to the server named RemoteASE by selecting the version string:

```
FORWARD TO RemoteASE {SELECT @@version};
```

Example 2

The following statements show a passthrough session with the server named RemoteASE:

```
FORWARD TO RemoteASE;
    SELECT * FROM titles;
    SELECT * FROM authors;
FORWARD TO;
```

Remote Procedure Calls (RPCs)

SAP Sybase IQ users can issue procedure calls to remote servers that support the feature.

SAP Sybase IQ, SQL Anywhere, and Adaptive Server, as well as Oracle and DB2, support this feature. Issuing a remote procedure call is similar to using a local procedure call.

Creating Remote Procedures

Administrators can create remote procedures in Interactive SQL.

Prerequisites

You must have the `MANAGE REPLICATION` system privilege.

Task

If a remote procedure can return a result set, even if it does not always return one, then the local procedure definition must contain a `RESULT` clause.

1. Connect to the host database.
2. Execute a statement to define the procedure. For example:

```
CREATE PROCEDURE RemoteWho()  
AT 'bostonase.master.dbo.sp_who'
```

This example specifies a parameter when calling a remote procedure:

```
CREATE PROCEDURE RemoteUser ( IN username CHAR( 30 ) )  
AT 'bostonase.master.dbo.sp_helpuser';  
CALL RemoteUser( 'joe' );
```

Remote Transactions

Transaction management involving remote servers uses a *two-phase commit* protocol.

SAP Sybase IQ implements a strategy that ensures transaction integrity for most scenarios.

Remote transaction management

The method for managing transactions involving remote servers uses a two-phase commit protocol. SAP Sybase IQ implements a strategy that ensures transaction integrity for most scenarios. However, when more than one remote server is invoked in a transaction, there is still a chance that a distributed unit of work will be left in an undetermined state. Even though two-phase commit protocol is used, no recovery process is included.

The general logic for managing a user transaction is as follows:

1. SAP Sybase IQ prefaces work to a remote server with a `BEGIN TRANSACTION` notification.
2. When the transaction is ready to be committed, SAP Sybase IQ sends a `PREPARE TRANSACTION` notification to each remote server that has been part of the transaction. This ensures that the remote server is ready to commit the transaction.
3. If a `PREPARE TRANSACTION` request fails, all remote servers are instructed to roll back the current transaction.

If all **PREPARE TRANSACTION** requests are successful, the server sends a **COMMIT TRANSACTION** request to each remote server involved with the transaction.

Any statement preceded by **BEGIN TRANSACTION** can begin a transaction. Other statements are sent to a remote server to be executed as a single, remote unit of work.

Remote Transaction Restrictions

Remote transaction management has savepoints and nested statement restrictions.

Restrictions on transaction management include:

- Savepoints are not propagated to remote servers.
- If nested **BEGIN TRANSACTION** and **COMMIT TRANSACTION** statements are included in a transaction that involves remote servers, only the outermost set of statements is processed. The inconsistent set, containing the **BEGIN TRANSACTION** and **COMMIT TRANSACTION** statements, is not transmitted to remote servers.

Internal Operations

This section describes the underlying steps that SAP Sybase IQ performs on remote servers on behalf of client applications.

Query Parsing

When a statement is received from a client, the database server parses it. The database server raises an error if the statement is not a valid SQL Anywhere SQL statement.

Query Normalization

In query normalization, referenced objects are verified and data type compatibility is checked.

For example, consider this query:

```
SELECT *  
FROM t1  
WHERE c1 = 10
```

The query normalization stage verifies that table `t1` with a column `c1` exists in the system tables. It also verifies that the data type of column `c1` is compatible with the value 10. If the column's data type is `DATETIME`, for example, this statement is rejected.

Query preprocessing

Query preprocessing prepares the query for optimization. It may change the representation of a statement so that the SQL statement that SAP Sybase IQ generates for passing to a remote

server is syntactically different from the original statement, even though it is semantically equivalent.

Preprocessing performs view expansion so that a query can operate on tables referenced by the view. Expressions may be reordered and subqueries may be transformed to improve processing efficiency. For example, some subqueries may be converted into joins.

Complete passthrough of the statement

For efficiency, SAP Sybase IQ passes off as much of the statement as possible to the remote server. Often, this is the complete statement originally given to SAP Sybase IQ.

SAP Sybase IQ hands off the complete statement when:

- Every table in the statement resides on the same remote server.
- The remote server can process all of the syntax in the statement.

In rare conditions, it may actually be more efficient to let SAP Sybase IQ do some of the work instead of the remote server doing it. For example, SAP Sybase IQ may have a better sorting algorithm. In this case, you may consider altering the capabilities of a remote server using the ALTER SERVER statement.

Partial passthrough of the statement

If a statement contains references to multiple servers, or uses SQL features not supported by a remote server, the query is decomposed into simpler parts.

SELECT

SELECT statements are broken down by removing portions that cannot be passed on and letting SAP Sybase IQ perform the work. For example, suppose a remote server cannot process the ATAN2 function in the following statement:

```
SELECT a,b,c
WHERE ATAN2 ( b, 10 ) > 3
AND c = 10;
```

The statement sent to the remote server would be converted to:

```
SELECT a,b,c WHERE c = 10;
```

Then, SAP Sybase IQ locally applies WHERE ATAN2 (b, 10) > 3 to the intermediate result set.

Joins

When a statement contains joins between tables in multiple locations, IQ will attempt to push joins of collocated tables to the server on which they reside. The results of that join will then be joined by IQ with results from other remote tables or local tables. IQ will always prefer to push as much join work as is possible to remote servers. When IQ joins remote tables with local IQ tables, IQ may choose to use any join algorithm it supports.

The choice of algorithm is based on cost estimates. These algorithms can include nested loop, hash, or sort-merge joins.

When a nested loop join is chosen between an IQ and a remote table, every effort is made to make the remote table the outermost table in the join. This is due to the high cost of network I/O that makes look-ups against a remote table usually much higher than a local table.

UPDATE and DELETE

When a qualifying row is found, if SAP Sybase IQ cannot pass off an UPDATE or DELETE statement entirely to a remote server, it must change the statement into a table scan containing as much of the original WHERE clause as possible, followed by a positioned UPDATE or DELETE statement that specifies WHERE CURRENT OF *cursor-name*.

For example, when the function ATAN2 is not supported by a remote server:

```
UPDATE t1
SET a = atan2( b, 10 )
WHERE b > 5;
```

Would be converted to the following:

```
SELECT a, b
FROM t1
WHERE b > 5;
```

Each time a row is found, SAP Sybase IQ would calculate the new value of a and execute:

```
UPDATE t1
SET a = 'new value'
WHERE CURRENT OF CURSOR;
```

If a already has a value that equals the new value, a positioned UPDATE would not be necessary, and would not be sent remotely.

To process an UPDATE or DELETE statement that requires a table scan, the remote data source must support the ability to perform a positioned UPDATE or DELETE (WHERE CURRENT OF *cursor-name*). Some data sources do not support this capability.

Note: Temporary tables cannot be updated

An UPDATE or DELETE cannot be performed if an intermediate temporary table is required. This occurs in queries with ORDER BY and some queries with subqueries.

Remote Data Access Troubleshooting

This section provides some suggestions for troubleshooting access to remote servers.

Features not supported for remote data

The following SAP Sybase IQ features are not supported on remote data:

- ALTER TABLE statement on remote tables.
- triggers defined on proxy tables.
- foreign keys that refer to remote tables.
- READTEXT, WRITETEXT, and TEXTPTR functions.
- positioned UPDATE and DELETE statements.
- UPDATE and DELETE statements requiring an intermediate temporary table.
- backward scrolling on cursors opened against remote data. Fetch statements must be NEXT or RELATIVE 1.
- calls to functions that contain an expression that references a proxy table.
- If a column on a remote table has a name that is a keyword on the remote server, you cannot access data in that column. You can execute a CREATE EXISTING TABLE statement, and import the definition but you cannot select that column.

Case sensitivity

The case sensitivity setting of your SAP Sybase IQ database should match the settings used by any remote servers accessed.

SAP Sybase IQ databases are created case insensitive by default. With this configuration, unpredictable results may occur when selecting from a case-sensitive database. Different results will occur depending on whether ORDER BY or string comparisons are pushed off to a remote server, or evaluated by the local SAP Sybase IQ server.

Connectivity tests

Take the following steps to ensure that you can connect to a remote server:

- Make sure that you can connect to a remote server using a client tool such as Interactive SQL before configuring SAP Sybase IQ.
- Perform a simple passthrough statement to a remote server to check your connectivity and remote login configuration. For example:

```
FORWARD TO RemoteSA {SELECT @@version};
```

- Turn on remote tracing for a trace of the interactions with remote servers. For example:
SET OPTION cis_option = 7;

Once you have turned on remote tracing, the tracing information appears in the database server messages window. You can log this output to a file by specifying the -o server option when you start the database server.

Remote data access connections via ODBC

If you access remote databases via ODBC, the connection to the remote server is given a name. You can use the name to drop the connection to cancel a remote request.

The connections are named `ASACIS_conn-name`, where *conn-name* is the connection ID of the local connection. The connection ID can be obtained from the `sa_conn_info` stored procedure.

Remote data access on multiplex servers

When you access a new proxy server on a secondary server, a timing problem may cause emergency server shutdown.

If emergency server shutdown occurs, reconnect to the server, or wait for some time and start a new transaction before trying to use the new proxy table.

Appendix: SQL Reference

Reference material for the SQL statements used in tasks in this document.

ALTER SERVER Statement

Modifies the attributes of a remote server. Changes made by **ALTER SERVER** do not take effect until the next connection to the remote server.

Quick Links:

Go to Parameters on page 961

Go to Examples on page 963

Go to Usage on page 963

Go to Standards on page 963

Go to Permissions on page 963

Syntax

```
ALTER SERVER server-name
  [ CLASS 'server-class' ]
  [ USING 'connection-info' ]
  [ CAPABILITY 'cap-name' { ON | OFF } ]
  [ CONNECTION CLOSE [ CURRENT | ALL | connection-id ] ]
```

server-class - (*back to Syntax*)

```
{ ASAJDBC
  | ASEJDBC
  | SAODBC
  | ASEODBC
  | DB2ODBC
  | MSSODBC
  | ORAODBC
  | ODBC }
```

connection-info - (*back to Syntax*)

```
{ machine-name:port-number [ /dbname ] | data-source-name }
```

Parameters

(*back to top*) on page 961

- **cap-name** – the name of a server capability
- **CLASS** – changes the server class.

- **USING** – if a JDBC-based server class is used, the USING clause is *hostname:port-number [/dbname]* where:
 - **hostname** – the machine on which the remote server runs.
 - **portnumber** – the TCP/IP port number on which the remote server listens. The default port number for SAP Sybase IQ and SAP Sybase SQL Anywhere® is 2638.
 - **dbname** – for SQL Anywhere remote servers, if you do not specify a *dbname*, the default database is used. For Adaptive Server, the default is the master database, and an alternative to using *dbname* is to another database by some other means (for example, in the **FORWARD TO** statement).

If an ODBC-based server class is used, the USING clause is the *data-source-name*, which is the ODBC Data Source Name.

- **CAPABILITY** – turns a server capability ON or OFF. Server capabilities are stored in the system table SYSCAPABILITY. The names of these capabilities are stored in the system table SYSCAPABILITYNAME. The SYSCAPABILITY table contains no entries for a remote server until the first connection is made to that server. At the first connection, SAP Sybase IQ interrogates the server about its capabilities and then populates SYSCAPABILITY. For subsequent connections, the server's capabilities are obtained from this table.

In general, you need not alter a server's capabilities. It might be necessary to alter capabilities of a generic server of class ODBC.

- **CONNECTION CLOSE** – when a user creates a connection to a remote server, the remote connection is not closed until the user disconnects from the local database. The CONNECTION CLOSE clause allows you to explicitly close connections to a remote server. You may find this useful when a remote connection becomes inactive or is no longer needed.

These SQL statements are equivalent and close the current connection to the remote server:

```
ALTER SERVER server-name CONNECTION CLOSE
ALTER SERVER server-name CONNECTION CLOSE CURRENT
```

You can close both ODBC and JDBC connections to a remote server using this syntax. You do not need the SERVER OPERATOR system privilege to execute either of these statements.

You can also disconnect a specific remote ODBC connection by specifying a connection ID, or disconnect all remote ODBC connections by specifying the ALL keyword. If you attempt to close a JDBC connection by specifying the connection ID or the ALL keyword, an error occurs. When the connection identified by *connection-id* is not the current local connection, the user must have the SERVER OPERATOR system privilege to be able to close the connection.

Examples

(back to top) on page 961

- **Example 1** – changes the server class of the Adaptive Server server named `ase_prod` so its connection to SAP Sybase IQ is ODBC-based. The Data Source Name is `ase_prod`.

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_prod'
```

- **Example 2** – changes a capability of server `infodc`:

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF
```

- **Example 3** – closes all connections to the remote server named `rem_test`:

```
ALTER SERVER rem_test
CONNECTION CLOSE ALL
```

- **Example 4** – closes the connection to the remote server named `rem_test` that has the connection ID 142536:

```
ALTER SERVER rem_test
CONNECTION CLOSE 142536
```

Usage

(back to top) on page 961

Side effects:

- Automatic commit

Standards

(back to top) on page 961

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- SAP Sybase Database product—Supported by Open Client/Open Server.

Permissions

(back to top) on page 961

Requires the SERVER OPERATOR system privilege.

CREATE EXISTING TABLE Statement

Creates a new proxy table that represents an existing table on a remote server.

Quick Links:

Go to Parameters on page 964

Go to Examples on page 965

Go to Usage on page 965

Go to Standards on page 966

Go to Permissions on page 966

Syntax

```
CREATE EXISTING TABLE [owner.]table_name
  [ ( column-definition, ... ) ]
  AT 'location-string'
```

column-definition - (*back to Syntax*)
 column-name data-type [NOT NULL]

location-string - (*back to Syntax*)
 remote-server-name.[db-name].[owner].object-name | remote-server-
 name; [db-name]; [owner]; object-name

Parameters

(*back to top*) on page 964

- **column-definition** – if you do not specify column definitions, SAP Sybase IQ derives the column list from the metadata it obtains from the remote table. If you do specify column definitions, SAP Sybase IQ verifies them. When SAP Sybase IQ checks column names, data types, lengths, and null properties:
 - Column names must match identically (although case is ignored).
 - Data types in **CREATE EXISTING TABLE** must match or be convertible to the data types of the column on the remote location. For example, a local column data type is defined as NUMERIC, whereas the remote column data type is MONEY. You may encounter some errors, if you select from a table in which the data types do not match or other inconsistencies exist.
 - Each column's NULL property is checked. If the local column's NULL property is not identical to the remote column's NULL property, a warning message is issued, but the statement is not aborted.
 - Each column's length is checked. If the lengths of CHAR, VARCHAR, BINARY, DECIMAL, and NUMERIC columns do not match, a warning message is issued, but the

command is not aborted. You might choose to include only a subset of the actual remote column list in your **CREATE EXISTING** statement.

- **AT** – specifies the location of the remote object. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the location string, the semicolon is the field delimiter. If no semicolon is present, a period is the field delimiter. This allows you to use file names and extensions in the database and owner fields. Semicolon field delimiters are used primarily with server classes that are not currently supported; however, you can also use them where a period would also work as a field delimiter.

For example, this statement maps the table `proxy_a1` to the SQL Anywhere database `mydb` on the remote server `myasa`:

```
CREATE EXISTING TABLE
proxy_a1
AT 'myasa;mydb;;a1'
```

Examples

(back to top) on page 964

- **Example 1** – create a proxy table named `nation` for the `nation` table at the remote server `server_a`:

```
CREATE EXISTING TABLE nation
( n_nationkey int,
  n_name char(25),
  n_regionkey int,
  n_comment char(152))
AT 'server_a.dbl.joe.nation'
```

- **Example 2** – create a proxy table named `blurbs` for the `blurbs` table at the remote server `server_a`. SAP Sybase IQ derives the column list from the metadata it obtains from the remote table:

```
CREATE EXISTING TABLE blurbs
AT 'server_a.dbl.joe.blurbs'
```

- **Example 3** – create a proxy table named `rda_employee` for the `Employees` table at the SAP Sybase IQ remote server `remote_iqdemo_srv`:

```
CREATE EXISTING TABLE rda_employee
AT 'remote_iqdemo_srv..dba.Employees'
```

Usage

(back to top) on page 964

CREATE EXISTING TABLE is a variant of the **CREATE TABLE** statement. The **EXISTING** keyword is used with **CREATE TABLE** to specify that a table already exists remotely, and that its metadata is to be imported into SAP Sybase IQ. This establishes the remote table as a

visible entity to its users. SAP Sybase IQ verifies that the table exists at the external location before it creates the table.

Tables used as proxy tables cannot have names longer than 30 characters.

If the object does not exist (either as a host data file or remote server object), the statement is rejected with an error message.

Index information from the host data file or remote server table is extracted and used to create rows for the system table `sysindexes`. This defines indexes and keys in server terms and enables the query optimizer to consider any indexes that might exist on this table.

Referential constraints are passed to the remote location when appropriate.

In a simplex environment, you cannot create a proxy table that refers to a remote table on the same node. In a multiplex environment, you cannot create a proxy table that refers to the remote table defined within the multiplex.

For example, in a simplex environment, if you try to create proxy table `proxy_e`, which refers to base table `Employees` defined on the same node, the **CREATE EXISTING TABLE** statement is rejected with an error message. In a multiplex environment, the **CREATE EXISTING TABLE** statement is rejected if you create proxy table `proxy_e` from any node (coordinator or secondary) that refers to remote table `Employees` defined within a multiplex.

Standards

(back to top) on page 964

- SQL—ISO/ANSI SQL compliant.
- SAP Sybase Database product—Supported by Open Client/Open Server.

Permissions

(back to top) on page 964

For table to be owned by self – Requires one of:

- CREATE ANY TABLE system privilege.
- CREATE ANY OBJECT system privilege.

For table to be owned by any user – Requires the CREATE ANY TABLE system privilege.

CREATE SERVER Statement

Adds a server to the `ISYSSERVER` table.

Quick Links:

Go to Parameters on page 967

Go to Examples on page 967

Go to Usage on page 968

Go to Standards on page 968

Go to Permissions on page 968

Syntax

```
CREATE SERVER server-name
  CLASS 'server-class'
  USING 'connection-info'
  [ READ ONLY ]
```

server-class - (*back to Syntax*)

```
{ ASAJDBC
  | ASEJDBC
  | SAODBC
  | ASEODBC
  | DB2ODBC
  | MSSODBC
  | ORODBC
  | ODBC }
```

connection-info - (*back to Syntax*)

```
{ machine-name:port-number [ /dbname ] | data-source-name }
```

Parameters

(*back to top*) on page 966

- **USING** – if a JDBC-based server class is used, the USING clause is *hostname:port-number* [/dbname] where:
 - **hostname** – the machine on which the remote server runs.
 - **portnumber** – the TCP/IP port number on which the remote server listens. The default port number for SAP Sybase IQ and SAP Sybase SQL Anywhere® is 2638.
 - **dbname** – for SQL Anywhere remote servers, if you do not specify a *dbname*, the default database is used. For Adaptive Server, the default is the master database, and an alternative to using *dbname* is to another database by some other means (for example, in the **FORWARD TO** statement).

If an ODBC-based server class is used, the USING clause is the *data-source-name*, which is the ODBC Data Source Name.

- **READ ONLY** – specifies that the remote server is a read-only data source. Any update request is rejected by SAP Sybase IQ.

Examples

(*back to top*) on page 966

Appendix: SQL Reference

- **Example 1** – create a remote server for the JDBC-based Adaptive Server server named `ase_prod`. Its machine name is “banana” and port number is 3025.

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025'
```

- **Example 2** – create an SQL Anywhere remote server named `testasa` on the machine “apple” with listening on port number 2638:

```
CREATE SERVER testasa
CLASS 'asajdbc'
USING 'apple:2638'
```

- **Example 3** – create a remote server for the Oracle server named `oracle723`. Its ODBC Data Source Name is “oracle723”:

```
CREATE SERVER oracle723
CLASS 'oraodbc'
USING 'oracle723'
```

Usage

(back to top) on page 966

CREATE SERVER defines a remote server from the SAP Sybase IQ catalogs.

Side Effects

- Automatic commit

Standards

(back to top) on page 966

- SQL—ISO/ANSI SQL compliant.
- SAP Sybase Database product—Supported by Open Client/Open Server.

Permissions

(back to top) on page 966

Requires the `SERVER OPERATOR` system privilege.

CREATE TABLE Statement

Creates a new table in the database or on a remote server.

Quick Links:

Go to Parameters on page 970

Go to Examples on page 980

Go to *Usage* on page 982

Go to *Standards* on page 984

Go to *Permissions* on page 984

Syntax

```
CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE
  [ IF NOT EXISTS ] [ owner. ] table-name
  ... ( column-definition [ column-constraint ] ...
  [ , column-definition [ column-constraint ] ... ]
  [ , table-constraint ] ... )
  [ { ENABLE | DISABLE } RLV STORE

  ... [ IN dbspace-name ]
  ... [ ON COMMIT { DELETE | PRESERVE } ROWS ]
  [ AT location-string ]
  [ PARTITION BY
    range-partitioning-scheme
    | hash-partitioning-scheme
    | composite-partitioning-scheme ]

column-definition - (back to Syntax)
  column-name data-type
  [ [ NOT ] NULL ]
  [ DEFAULT default-value | IDENTITY ]
  [ PARTITION | SUBPARTITION ( partition-name IN dbspace-name
  [ , ... ] ) ]

default-value - (back to column-definition)
  special-value
  | string
  | global variable
  | [ - ] number
  | ( constant-expression )
  | built-in-function( constant-expression )
  | AUTOINCREMENT
  | CURRENT DATABASE
  | CURRENT REMOTE USER
  | NULL
  | TIMESTAMP
  | LAST USER

special-value - (back to default value)
  CURRENT
  { DATE
  | TIME
  | TIMESTAMP
  | USER
  | PUBLISHER }
  | USER

column-constraint - (back to Syntax)
  [ CONSTRAINT constraint-name ] {
  { UNIQUE
```

```

| PRIMARY KEY
| REFERENCES table-name [ ( column-name ) ] [ action ]
}
[ IN dbspace-name ]
| CHECK ( condition )
| IQ UNIQUE ( integer )
}

table-constraint - (back to Syntax)
[ CONSTRAINT constraint-name ]
{ { UNIQUE ( column-name [ , column-name ] ... )
  | PRIMARY KEY ( column-name [ , column-name ] ... )
  }
[ IN dbspace-name ]
| foreign-key-constraint
| CHECK ( condition )
| IQ UNIQUE ( integer )
}

foreign-key-constraint - (back to table-constraint)
FOREIGN KEY [ role-name ] [ ( column-name [ , column-name ] ... ) ]
...REFERENCES table-name [ ( column-name [ , column-name ] ... ) ]
...[ actions ] [ IN dbspace-name ]

actions - (back to foreign-key-constraint)
[ ON { UPDATE | DELETE } RESTRICT ]

location-string - (back to Syntax) or (back to composite-partitioning-
scheme)
{ remote-server-name. [ db-name ].[ owner ].object-name
  | remote-server-name; [ db-name ]; [ owner ];object-name }

range-partitioning-scheme - (back to Syntax)
RANGE ( partition-key ) ( range-partition-decl [ , range-partition-decl ... ] )

partition-key - (back to range-partitioning-scheme) or (back to hash-
partitioning-scheme)
column-name

range-partition-decl - (back to range-partitioning-scheme)
VALUES <= ( { constant-expr
  | MAX } [ , { constant-expr
  | MAX } ] ... )
[ IN dbspace-name ]

hash-partitioning-scheme - (back to Syntax) or (back to composite-
partitioning-scheme)
HASH ( partition-key [ , partition-key, ... ] )

composite-partitioning-scheme - (back to Syntax)
hash-partitioning-scheme SUBPARTITION range-partitioning-scheme

```

Parameters

(back to top) on page 968

- **IN** – used in the column-definition, column-constraint, table-constraint, foreign-key, and partition-decl clauses to specify the dbspace where the object is to be created. If the IN clause is omitted, SAP Sybase IQ creates the object in the dbspace where the table is assigned.

Specify **SYSTEM** with this clause to put either a permanent or temporary table in the catalog store. Specify **IQ_SYSTEM_TEMP** to store temporary user objects (tables, partitions, or table indexes) in **IQ_SYSTEM_TEMP** or, if the **TEMP_DATA_IN_SHARED_TEMP** option is set 'ON', and the **IQ_SHARED_TEMP** dbspace contains RW files, in **IQ_SHARED_TEMP**. (You cannot specify the IN clause with **IQ_SHARED_TEMP**.) All other use of the IN clause is ignored. By default, all permanent tables are placed in the main IQ store, and all temporary tables are placed in the temporary IQ store. Global temporary and local temporary tables can never be in the IQ store.

The following syntax is unsupported:

```
CREATE LOCAL TEMPORARY TABLE tabl(c1 int) IN IQ_SHARED_TEMP
```

A **BIT** data type column cannot be explicitly placed in a dbspace. The following is not supported for **BIT** data types:

```
CREATE TABLE t1(c1_bit bit IN iq_main);
```

- **ON COMMIT** – allowed for temporary tables only. By default, the rows of a temporary table are deleted on **COMMIT**.
- **AT** – creates a proxy table that maps to a remote location specified by the location-string clause. Proxy table names must be 30 characters or less. The **AT** clause supports semicolon (;) delimiters. If a semicolon is present anywhere in the location-string clause, the semicolon is the field delimiter. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields.

Semicolon field delimiters are used primarily with server classes not currently supported; however, you can also use them in situations where a period would also work as a field delimiter. For example, this statement maps the table `proxy_a` to the SQL Anywhere database `mydb` on the remote server `myasa`:

```
CREATE TABLE proxy_a1
AT 'myasa;mydb;;a1'
```

Foreign-key definitions are ignored on remote tables. Foreign-key definitions on local tables that refer to remote tables are also ignored. Primary key definitions are sent to the remote server if the server supports primary keys.

In a simplex environment, you cannot create a proxy table that refers to a remote table on the same node. In a multiplex environment, you cannot create a proxy table that refers to the remote table defined within the multiplex.

- **IF NOT EXISTS** – if the named object already exists, no changes are made and an error is not returned.
- **{ ENABLE | DISABLE } RLV STORE** – registers this table with the RLV store for real-time in-memory updates. Not supported for IQ temporary tables. This value overrides the value of the database option **BASE_TABLES_IN_RLV**. Requires the CREATE TABLE system privilege and CREATE permissions on the RLV store dbspace to set this value to ENABLE.
- **column-definition** – defines a table column. Allowable data types are described in *Reference: Building Blocks, Tables, and Procedures >SQL Data Types*. Two columns in the same table cannot have the same name. You can create up to 45,000 columns; however, there might be performance penalties in tables with more than 10,000 columns.
 - **[NOT] NULL]** – includes or excludes NULL values. If NOT NULL is specified, or if the column is in a UNIQUE or PRIMARY KEY constraint, the column cannot contain any NULL values. The limit on the number of columns per table that allow NULLs is approximately $8 * (\text{database-page-size} - 30)$.
 - **DEFAULT default-value** – specify a default column value with the DEFAULT keyword in the CREATE TABLE (and ALTER TABLE) statement. A DEFAULT value is used as the value of the column in any INSERT (or LOAD) statement that does not specify a column value.
 - **DEFAULT AUTOINCREMENT** – the value of the DEFAULT AUTOINCREMENT column uniquely identifies every row in a table. Columns of this type are also known as IDENTITY columns, for compatibility with Adaptive Server. The IDENTITY/DEFAULT AUTOINCREMENT column stores sequential numbers that are automatically generated during inserts and updates. When using IDENTITY or DEFAULT AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type, with scale 0. The column value might also be NULL. You must qualify the specified table name with the owner name.

ON inserts into the table. If a value is not specified for the IDENTITY/DEFAULT AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column, it is used; if the specified value is not larger than the current maximum value for the column, that value is used as a starting point for subsequent inserts.

Deleting rows does not decrement the IDENTITY/AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. The database option IDENTITY_INSERT must be set to the table name to perform an insert into an IDENTITY/AUTOINCREMENT column.

For example, this creates a table with an IDENTITY column and explicitly adds some data to it:

```
CREATE TABLE mytable(c1 INT IDENTITY);
SET TEMPORARY OPTION IDENTITY_INSERT = "DBA".mytable;
INSERT INTO mytable VALUES(5);
```

After an explicit insert of a row number less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

- **IDENTITY** – a Transact-SQL® -compatible alternative to using the AUTOINCREMENT default. In SAP Sybase IQ, the identity column may be created using either the IDENTITY or the DEFAULT AUTOINCREMENT clause.
- **table-constraint** – helps ensure the integrity of data in the database. There are four types of integrity constraints:
 - **UNIQUE** – identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named columns. A table may have more than one unique constraint.
 - **PRIMARY KEY** – the same as a UNIQUE constraint except that a table can have only one primary-key constraint. You cannot specify the PRIMARY KEY and UNIQUE constraints for the same column. The primary key usually identifies the best identifier for a row. For example, the customer number might be the primary key for the customer table.
 - **FOREIGN KEY** – restricts the values for a set of columns to match the values in a primary key or uniqueness constraint of another table. For example, a foreign-key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the customer table.

You cannot create foreign-key constraints on local temporary tables. Global temporary tables must be created with ON COMMIT PRESERVE ROWS.

- **CHECK** – allows arbitrary conditions to be verified. For example, a check constraint could be used to ensure that a column called Gender contains only the values male or female. No row in a table is allowed to violate a constraint. If an INSERT or UPDATE statement would cause a row to violate a constraint, the operation is not permitted and the effects of the statement are undone.

Column identifiers in column check constraints that start with the symbol '@' are placeholders for the actual column name. A statement of the form:

```
CREATE TABLE t1 (c1 INTEGER CHECK (@foo < 5))
```

is exactly the same as this statement:

```
CREATE TABLE t1 (c1 INTEGER CHECK (c1 < 5))
```

Column identifiers appearing in table check constraints that start with the symbol '@' are not placeholders.

If a statement would cause changes to the database that violate an integrity constraint, the statement is effectively not executed and an error is reported. (Effectively means that any changes made by the statement before the error was detected are undone.)

SAP Sybase IQ enforces single-column UNIQUE constraints by creating an HG index for that column.

Note: You cannot define a column with a BIT data type as a UNIQUE or PRIMARY KEY constraint. Also, the default for columns of BIT data type is to not allow NULL values; you can change this by explicitly defining the column as allowing NULL values.

- **column-constraint** – restricts the values the column can hold. Column and table constraints help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported. Column constraints are abbreviations for the corresponding table constraints. For example, these are equivalent:

```
CREATE TABLE Products (
    product_num integer UNIQUE
)
CREATE TABLE Products (
    product_num integer,
    UNIQUE ( product_num )
)
```

Column constraints are normally used unless the constraint references more than one column in the table. In these cases, a table constraint must be used.

- **IQ UNIQUE** – defines the expected cardinality of a column and determines whether the column loads as Flat FP or NBit FP. An IQ UNIQUE(n) value explicitly set to 0 loads the column as Flat FP. Columns without an IQ UNIQUE constraint implicitly load as NBit up to the limits defined by the FP_NBIT_AUTOSIZE_LIMIT, FP_NBIT_LOOKUP_MB, and FP_NBIT_ROLLOVER_MAX_MB options:
 - FP_NBIT_AUTOSIZE_LIMIT limits the number of distinct values that load as NBit
 - FP_NBIT_LOOKUP_MB sets a threshold for the total NBit dictionary size
 - FP_NBIT_ROLLOVER_MAX_MB sets the dictionary size for implicit NBit rollovers from NBit to Flat FP
 - FP_NBIT_ENFORCE_LIMITS enforces NBit dictionary sizing limits. This option is OFF by default

Using IQ UNIQUE with an n value less than the FP_NBIT_AUTOSIZE_LIMIT is not necessary. Auto-size functionality automatically sizes all low or medium cardinality columns as NBit. Use IQ UNIQUE in cases where you want to load the column as Flat FP or when you want to load a column as NBit when the number of distinct values exceeds the FP_NBIT_AUTOSIZE_LIMIT.

Note:

- Consider memory usage when specifying high IQ UNIQUE values. If machine resources are limited, avoid loads with FP_NBIT_ENFORCE_LIMITS='OFF' (default).

Prior to SAP Sybase IQ 16.0, an IQ UNIQUE *n* value > 16777216 would rollover to Flat FP. In 16.0, larger IQ UNIQUE values are supported for tokenization, but may require significant memory resource requirements depending on cardinality and column width.

- BIT, BLOB, and CLOB data types do not support NBit dictionary compression. If FP_NBIT_IQ15_COMPATIBILITY='OFF', a non-zero IQ UNIQUE column specification in a CREATE TABLE or ALTER TABLE statement that includes these data types returns an error.

- **column-constraint and table-constraint clauses** – column and table constraints help ensure the integrity of data in the database.
 - **PRIMARY KEY or PRIMARY KEY (column-name, ...)** – the primary key for the table consists of the listed columns, and none of the named columns can contain any NULL values. SAP Sybase IQ ensures that each row in the table has a unique primary key value. A table can have only one PRIMARY KEY.

When the second form is used (PRIMARY KEY followed by a list of columns), the primary key is created including the columns in the order in which they are defined, not the order in which they are listed.

When a column is designated as PRIMARY KEY, FOREIGN KEY, or UNIQUE, SAP Sybase IQ creates a High_Group index for it automatically. For multicolumn primary keys, this index is on the primary key, not the individual columns. For best performance, you should also index each column with a HG or LF index separately.

- **REFERENCES primary-table-name [(primary-column-name)]** – defines the column as a foreign key for a primary key or a unique constraint of a primary table. Normally, a foreign key would be for a primary key rather than an unique constraint. If a primary column name is specified, it must match a column in the primary table which is subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. Otherwise the foreign key references the primary key of the second table. Primary key and foreign key must have the same data type and the same precision, scale, and sign. Only a non unique single-column HG index is created for a single-column foreign key. For a multicolumn foreign key, SAP Sybase IQ creates a non unique composite HG index. The maximum width of a multicolumn composite key for a unique or non unique HG index is 1KB.

A temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a temporary table. Local temporary tables cannot have or be referenced by a foreign key.

- **FOREIGN KEY [role-name] [(...)] REFERENCES primary-table-name [(...)]** – defines foreign-key references to a primary key or a unique constraint in another table. Normally, a foreign key would be for a primary key rather than an unique constraint. (In this description, this other table is called the primary table.)

If the primary table column names are not specified, the primary table columns are the columns in the table's primary key. If foreign key column names are not specified, the

foreign-key columns have the same names as the columns in the primary table. If foreign-key column names are specified, then the primary key column names must be specified, and the column names are paired according to position in the lists.

If the primary table is not the same as the foreign-key table, either the unique or primary key constraint must have been defined on the referenced key. Both referenced key and foreign key must have the same number of columns, of identical data type with the same sign, precision, and scale.

The value of the row's foreign key must appear as a candidate key value in one of the primary table's rows unless one or more of the columns in the foreign key contains nulls in a null allows foreign key column.

Any foreign-key column not explicitly defined is automatically created with the same data type as the corresponding column in the primary table. These automatically created columns cannot be part of the primary key of the foreign table. Thus, a column used in both a primary key and foreign key must be explicitly created.

role-name is the name of the foreign key. The main function of *role-name* is to distinguish two foreign keys to the same table. If no *role-name* is specified, the role name is assigned as follows:

1. If there is no foreign key with a *role-name* the same as the table name, the table name is assigned as the *role-name*.
2. If the table name is already taken, the *role-name* is the table name concatenated with a zero-padded 3-digit number unique to the table.

The referential integrity action defines the action to be taken to maintain foreign-key relationships in the database. Whenever a primary key value is changed or deleted from a database table, there may be corresponding foreign key values in other tables that should be modified in some way. You can specify an ON DELETE clause, followed by the RESTRICT clause.

- **RESTRICT** – generates an error if you try to update or delete a primary key value while there are corresponding foreign keys elsewhere in the database. Generates an error if you try to update a foreign key so that you create new values unmatched by a candidate key. This is the default action, unless you specify that LOAD optionally reject rows that violate referential integrity. This enforces referential integrity at the statement level.

If you use CHECK ON COMMIT without specifying any actions, then RESTRICT is implied as an action for DELETE. SAP Sybase IQ does not support CHECK ON COMMIT.

a global temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a global temporary table. Local temporary tables cannot have or be referenced by a foreign key.

- **CHECK (condition)** – no row is allowed to fail the condition. If an INSERT statement would cause a row to fail the condition, the operation is not permitted and the effects of the statement are undone.

The change is rejected only if the condition is FALSE; in particular, the change is allowed if the condition is UNKNOWN. CHECK condition is not enforced by SAP Sybase IQ.

Note: If possible, do not define referential integrity foreign key-primary key relationships in SAP Sybase IQ unless you are certain there are no orphan foreign keys.

- **Remote Tables** – foreign-key definitions are ignored on remote tables. Foreign-key definitions on local tables that refer to remote tables are also ignored. Primary-key definitions are sent to the remote server if the server supports it.
- **PARTITION BY** – divides large tables into smaller, more manageable storage objects. Partitions share the same logical attributes of the parent table, but can be placed in separate dbspaces and managed individually. SAP Sybase IQ supports several table partitioning schemes:
 - hash-partitions
 - range-partitions
 - composite-partitions

A partition-key is the column or columns that contain the table partitioning keys. Partition keys can contain NULL and DEFAULT values, but cannot contain:

- LOB (BLOB or CLOB) columns
- BINARY, or VARBINARY columns
- CHAR or VARCHAR columns whose length is over 255 bytes
- BIT columns
- FLOAT/DOUBLE/REAL columns
- **PARTITION BY RANGE** – partitions rows by a range of values in the partitioning column. Range partitioning is restricted to a single partition key column and a maximum of 1024 partitions. In a range-partitioning-scheme, the partition-key is the column that contains the table partitioning keys:

```
range-partition-decl:
  partition-name VALUES <= ( {constant-expr | MAX } [ ,
  { constant-expr | MAX } ]... )
  [ IN dbspace-name ]
```

The partition-name is the name of a new partition on which table rows are stored. Partition names must be unique within the set of partitions on a table. The partition-name is required.

- **VALUE** – specifies the inclusive upper bound for each partition (in ascending order). The user must specify the partitioning criteria for each range partition to guarantee that

each row is distributed to only one partition. NULLs are allowed for the partition column and rows with NULL as partition key value belong to the first table partition. However, NULL cannot be the bound value.

There is no lower bound (MIN value) for the first partition. Rows of NULL cells in the first column of the partition key will go to the first partition. For the last partition, you can either specify an inclusive upper bound or MAX. If the upper bound value for the last partition is not MAX, loading or inserting any row with partition key value larger than the upper bound value of the last partition generates an error.

- **Max** – denotes the infinite upper bound and can only be specified for the last partition.
- **IN** – specifies the dbspace in the partition-decl on which rows of the partition should reside.

These restrictions affect partitions keys and bound values for range partitioned tables:

- Partition bounds must be constants, not constant expressions.
- Partition bounds must be in ascending order according to the order in which the partitions were created. That is, the upper bound for the second partition must be higher than for the first partition, and so on.

In addition, partition bound values must be compatible with the corresponding partition-key column data type. For example, VARCHAR is compatible with CHAR.

- If a bound value has a different data type than that of its corresponding partition key column, SAP Sybase IQ converts the bound value to the data type of the partition key column, with these exceptions:
- Explicit conversions are not allowed. This example attempts an explicit conversion from INT to VARCHAR and generates an error:

```
CREATE TABLE Employees (emp_name VARCHAR(20))
PARTITION BY RANGE (emp_name)
(p1 VALUES <= (CAST (1 AS VARCHAR(20))),
p2 VALUES <= (CAST (10 AS VARCHAR(20)))
```

- Implicit conversions that result in data loss are not allowed. In this example, the partition bounds are not compatible with the partition key type. Rounding assumptions may lead to data loss and an error is generated:

```
CREATE TABLE emp_id (id INT) PARTITION BY RANGE (id) (p1 VALUES
<= (10.5), p2 VALUES <= (100.5))
```

- In this example, the partition bounds and the partition key data type are compatible. The bound values are directly converted to float values. No rounding is required, and conversion is supported:

```
CREATE TABLE id_emp (id FLOAT)
PARTITION BY RANGE (id) (p1 VALUES <= (10),
p2 VALUES <= (100))
```

- Conversions from non-binary data types to binary data types are not allowed. For example, this conversion is not allowed and returns an error:

```
CREATE TABLE newemp (name BINARY)
PARTITION BY RANGE (name)
```

```
(p1 VALUES <= ("Maarten"),
p2 VALUES <= ("Zymmerman"))
```

- NULL cannot be used as a boundary in a range-partitioned table.
- The row will be in the first partition if the cell value of the 1st column of the partition key evaluated to be NULL. SAP Sybase IQ supports only single column partition keys, so any NULL in the partition key distributes the row to the first partition.
- **PARTITION BY HASH** – maps data to partitions based on partition-key values processed by an internal hashing function. Hash partition keys are restricted to a maximum of eight columns with a combined declared column width of 5300 bytes or less. For hash partitions, the table creator determines only the partition key columns; the number and location of the partitions are determined internally.

In a hash-partitioning declaration, the partition-key is a column or group of columns, whose composite value determines the partition where each row of data is stored:

```
hash-partitioning-scheme:
HASH ( partition-key [ , partition-key, ... ] )
```

- **Restrictions** –
 - You can only hash partition a base table. Attempting to partitioning a global temporary table or a local temporary table raises an error.
 - You cannot add, drop, merge, or split a hash partition.
 - You cannot add or drop a column from a hash partition key.
- **PARTITION BY HASH RANGE** – subpartitions a hash-partitioned table by range. In a hash-range-partitioning-scheme declaration, a SUBPARTITION BY RANGE clause adds a new range subpartition to an existing hash-range partitioned table:

```
hash-range-partitioning-scheme:
PARTITION BY HASH ( partition-key [ , partition-key, ... ] )
  [ SUBPARTITION BY RANGE ( range-partition-decl [ , range-
partition-decl ... ] ) ]
```

The hash partition specifies how the data is logically distributed and colocated; the range subpartition specifies how the data is physically placed. The new range subpartition is logically partitioned by hash with the same hash partition keys as the existing hash-range partitioned table. The range subpartition key is restricted to one column.

- **Restrictions** –
 - You can only hash partition a base table. Attempting to partitioning a global temporary table or a local temporary table raises an error.
 - You cannot add, drop, merge, or split a hash partition.
 - You cannot add or drop a column from a hash partition key.

Note: Range-partitions and composite partitioning schemes, like hash-range partitions, require the separately licensed VLDB Management option.

Examples*(back to top)* on page 968

- **Example 1** – create a table named `SalesOrders2` with five columns. Data pages for columns `FinancialCode`, `OrderDate`, and `ID` are in `dbspace Dsp3`. Data pages for integer column `CustomerID` are in `dbspace Dsp1`. Data pages for CLOB column `History` are in `dbspace Dsp2`. Data pages for the primary key, `HG` for `ID`, are in `dbspace Dsp4`:

```
CREATE TABLE SalesOrders2 (
  FinancialCode CHAR(2),
  CustomerID int IN Dsp1,
  History CLOB IN Dsp2,
  OrderDate TIMESTAMP,
  ID BIGINT,
  PRIMARY KEY(ID) IN Dsp4
) IN Dsp3
```

- **Example 2** – create a table `fin_code2` with four columns. Data pages for columns `code`, `type`, and `id` are in the default `dbspace`, which is determined by the value of the database option `DEFAULT_DBSPACE`. Data pages for CLOB column `description` are in `dbspace Dsp2`. Data pages from foreign key `fk1`, `HG` for `c1` are in `dbspace Dsp4`:

```
CREATE TABLE fin_code2 (
  code INT,
  type CHAR(10),
  description CLOB IN Dsp2,
  id BIGINT,
  FOREIGN KEY fk1(id) REFERENCES SalesOrders(ID) IN Dsp4
)
```

- **Example 3** – create a table `t1` where partition `p1` is adjacent to `p2` and partition `p2` is adjacent to `p3`:

```
CREATE TABLE t1 (c1 INT, c2 INT)
PARTITION BY RANGE(c1)
(p1 VALUES <= (0), p2 VALUES <= (10), p3 VALUES <= (100))
```

- **Example 4** – create a RANGE partitioned table `bar` with six columns and three partitions, mapping data to partitions based on dates:

```
CREATE TABLE bar (
  c1 INT IQ UNIQUE(65500),
  c2 VARCHAR(20),
  c3 CLOB PARTITION (P1 IN Dsp11, P2 IN Dsp12,
  P3 IN Dsp13),
  c4 DATE,
  c5 BIGINT,
  c6 VARCHAR(500) PARTITION (P1 IN Dsp21,
  P2 IN Dsp22),
  PRIMARY KEY (c5) IN Dsp2) IN Dsp1
PARTITION BY RANGE (c4)
(P1 VALUES <= ('2006/03/31') IN Dsp31,
P2 VALUES <= ('2006/06/30') IN Dsp32,
```

```
P3 VALUES <= ('2006/09/30') IN Dsp33
) ;
```

Data page allocation for each partition:

Partition	Dbspaces	Columns
P1	Dsp31	c1, c2, c4, c5
P1	Dsp11	c3
P1	Dsp21	c6
P2	Dsp32	c1, c2, c4, c5
P2	Dsp12	c3
P2	Dsp22	c6
P3	Dsp33	c1, c2, c4, c5, c6
P3	Dsp13	c3
P1, P2, P3	Dsp1	lookup store of c1 and other shared data
P1, P2, P3	Dsp2	primary key (HG for c5)

- **Example 5**– create a HASH partitioned (table `tbl42`) that includes a PRIMARY KEY (column `c1`) and a HASH PARTITION KEY (columns `c4` and `c3`).

```
CREATE TABLE tbl42 (
  c1 BIGINT NOT NULL,
  c2 CHAR(2) IQ UNIQUE(50),
  c3 DATE IQ UNIQUE(36524),
  c4 VARCHAR(200),
  PRIMARY KEY (c1)
)
PARTITION BY HASH ( c4, c3 )
```

- **Example 6**– create a hash-ranged partitioned table with a PRIMARY KEY (column `c1`), a hash partition key (columns `c4` and `c2`) and a range subpartition key (column `c3`).

```
CREATE TABLE tbl42 (
  c1 BIGINT NOT NULL,
  c2 CHAR(2) IQ UNIQUE(50),
  c3 DATE,
  c4 VARCHAR(200),
  PRIMARY KEY (c1) IN Dsp1

  PARTITION BY HASH ( c4, c2 )
  SUBPARTITION BY RANGE ( c3 )
  ( P1 VALUES <= (2011/03/31) IN Dsp31,
```

```
P2 VALUES <= (2011/06/30) IN Dsp32,  
P3 VALUES <= (2011/09/30) IN Dsp33) ;
```

- **Example 7** – create a table for a library database to hold information on borrowed books:

```
CREATE TABLE borrowed_book (  
  date_borrowed DATE NOT NULL,  
  date_returned DATE,  
  book          CHAR(20)  
  REFERENCES library_books (isbn),  
  CHECK( date_returned >= date_borrowed )  
)
```

- **Example 8** – create table `t1` at the remote server `SERVER_A` and create a proxy table named `t1` that is mapped to the remote table:

```
CREATE TABLE t1  
( a INT,  
  b CHAR(10))  
AT 'SERVER_A.db1.joe.t1'
```

- **Example 9** – create table `tab1` that contains a column `c1` with a default value of the special constant `LAST USER`:

```
CREATE TABLE tab1(c1 CHAR(20) DEFAULT LAST USER)
```

- **Example 10** – create a local temporary table `tab1` that contains a column `c1`:

```
CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SYSTEM_TEMP
```

The example creates `tab1` in the `IQ_SYSTEM_TEMP` dbspace in the following cases:

- `DQP_ENABLED` logical server policy option is set `ON` but there are no read-write files in `IQ_SHARED_TEMP`
- `DQP_ENABLED` option is `OFF`, `TEMP_DATA_IN_SHARED_TEMP` logical server policy option is `ON`, but there are no read-write files in `IQ_SHARED_TEMP`
- Both the `DQP_ENABLED` option and the `TEMP_DATA_IN_SHARED_TEMP` option are set `OFF`

The example creates the same table `tab1` in the `IQ_SHARED_TEMP` dbspace in the following cases:

- `DQP_ENABLED` is `ON` and there are read-write files in `IQ_SHARED_TEMP`
- `DQP_ENABLED` is `OFF`, `TEMP_DATA_IN_SHARED_TEMP` is `ON`, and there are read-write files in `IQ_SHARED_TEMP`
- **Example 11** – create a table `tab1` that is enabled to use row-level versioning, and real-time storage in the in-memory RLV store.

```
CREATE TABLE tab1 ( c1 INT, c2 CHAR(25) ) ENABLE RLV STORE
```

Usage

(back to top) on page 968

You can create a table for another user by specifying an owner name. If `GLOBAL TEMPORARY` or `LOCAL TEMPORARY` is not specified, the table is referred to as a base table. Otherwise, the table is a temporary table.

A created global temporary table exists in the database like a base table and remains in the database until it is explicitly removed by a `DROP TABLE` statement. The rows in a temporary table are visible only to the connection that inserted the rows. Multiple connections from the same or different applications can use the same temporary table at the same time and each connection sees only its own rows. A given connection inherits the schema of a global temporary table as it exists when the connection first refers to the table. The rows of a temporary table are deleted when the connection ends.

When you create a local temporary table, omit the owner specification. If you specify an owner when creating a temporary table, for example, `CREATE TABLE dbo.#temp (col1 int)`, a base table is incorrectly created.

An attempt to create a base table or a global temporary table will fail, if a local temporary table of the same name exists on that connection, as the new table cannot be uniquely identified by `owner.table`.

You can, however, create a local temporary table with the same name as an existing base table or global temporary table. References to the table name access the local temporary table, as local temporary tables are resolved first.

For example, consider this sequence:

```
CREATE TABLE t1 (c1 int);
INSERT t1 VALUES (9);

CREATE LOCAL TEMPORARY TABLE t1 (c1 int);
INSERT t1 VALUES (8);

SELECT * FROM t1;
```

The result returned is 8. Any reference to `t1` refers to the local temporary table `t1` until the local temporary table is dropped by the connection.

In a procedure, use the `CREATE LOCAL TEMPORARY TABLE` statement, instead of the `DECLARE LOCAL TEMPORARY TABLE` statement, when you want to create a table that persists after the procedure completes. Local temporary tables created using the `CREATE LOCAL TEMPORARY TABLE` statement remain until they are either explicitly dropped, or until the connection closes.

Local temporary tables created in `IF` statements using `CREATE LOCAL TEMPORARY TABLE` also persist after the `IF` statement completes.

SAP Sybase IQ does not support the `CREATE TABLE ENCRYPTED` clause for table-level encryption of SAP Sybase IQ tables. However, the `CREATE TABLE ENCRYPTED` clause is supported for SQL Anywhere tables in an SAP Sybase IQ database.

Side Effects

- Automatic commit

Standards

(back to top) on page 968

- SQL–Vendor extension to ISO/ANSI SQL grammar.

These are vendor extensions:

- The { **IN** | **ON** } *dbspace-name* clause
- The **ON COMMIT** clause
- Some of the default values
- SAP Sybase Database product–Supported by Adaptive Server, with some differences.
 - **Temporary tables** – you can create a temporary table by preceding the table name in a **CREATE TABLE** statement with a pound sign (#). These temporary tables are SAP Sybase IQ declared temporary tables, which are available only in the current connection. For information about declared temporary tables, see *DECLARELOCAL TEMPORARY TABLE Statement*.
 - **Physical placement** – physical placement of a table is carried out differently in SAP Sybase IQ and in Adaptive Server. The **ON segment-name** clause supported by Adaptive Server is supported in SAP Sybase IQ, but *segment-name* refers to an IQ dbspace.
 - **Constraints** – SAP Sybase IQ does not support named constraints or named defaults, but does support user-defined data types that allow constraint and default definitions to be encapsulated in the data type definition. It also supports explicit defaults and CHECK conditions in the **CREATE TABLE** statement.
 - **NULL** – (default) by default, columns in Adaptive Server default to NOT NULL, whereas in SAP Sybase IQ the default setting is NULL, to allow NULL values. This setting can be controlled using the `ALLOW_NULLS_BY_DEFAULT` option. See *ALLOW_NULLS_BY_DEFAULT Option [TSQL]*. To make your data definition statements transferable, explicitly specify NULL or NOT NULL.

Permissions

(back to top) on page 968

Table Type	Privileges Required
Base table in the IQ main store	<p>Table owned by self – Requires CREATE privilege on the dbspace where the table is created. Also requires one of:</p> <ul style="list-style-type: none"> • CREATE TABLE system privilege. • CREATE ANY OBJECT system privilege. <p>Table owned by any user – Requires CREATE privilege on the dbspace where the table is created. Also requires one of:</p> <ul style="list-style-type: none"> • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege.
Global temporary table	<p>Table owned by self – Requires one of:</p> <ul style="list-style-type: none"> • CREATE TABLE system privilege. • CREATE ANY OBJECT system privilege. <p>Table owned by any user – Requires one of:</p> <ul style="list-style-type: none"> • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege.
Proxy table	<p>Table owned by self – Requires one of:</p> <ul style="list-style-type: none"> • CREATE PROXY TABLE system privilege. • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege. <p>Table owned by any user – Requires one of:</p> <ul style="list-style-type: none"> • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege.

DROP SERVER Statement

Drops a remote server from the SAP Sybase IQ system tables.

Quick Links:

Go to Examples on page 986

Go to Usage on page 986

Go to Standards on page 986

Go to Permissions on page 986

Syntax

```
DROP SERVER server-name
```

Examples

(back to top) on page 985

- **Example 1** – this example drops the server IQ_prod:

```
DROP SERVER iq_prod
```

Usage

(back to top) on page 985

Before **DROP SERVER** succeeds, you must drop all the proxy tables that have been defined for the remote server.

Side Effects

- Automatic commit

Standards

(back to top) on page 985

- SQL—ISO/ANSI SQL compliant.
- SAP Sybase Database product—Supported by Open Client/Open Server.

Permissions

(back to top) on page 985

Requires the SERVER OPERATOR system privilege.

Index

- _close_extfn
 - v4 API method 128
- _describe_extfn 25, 99
- _enter_state_extfn 99
- _fetch_block_extfn
 - v4 API method 127
- _fetch_into_extfn
 - v4 API method 126
- _finish_extfn 98
- _leave_state_extfn 99
- _open_extfn
 - v4 API method 126
- _rewind_extfn
 - v4 API method 128
- _start_extfn 97
- d option
 - SQL preprocessor utility (iqsqlpp) 411
- e option
 - SQL preprocessor utility (iqsqlpp) 411
- gn option
 - threads 372
- h option
 - SQL preprocessor utility (iqsqlpp) 411
- k option
 - SQL preprocessor utility (iqsqlpp) 411
- m option
 - SQL preprocessor utility (iqsqlpp) 411
- n option
 - SQL preprocessor utility (iqsqlpp) 411
- o option
 - SQL preprocessor utility (iqsqlpp) 411
- q option
 - SQL preprocessor utility (iqsqlpp) 411
- r option
 - SQL preprocessor utility (iqsqlpp) 411
- s option
 - SQL preprocessor utility (iqsqlpp) 411
- u option
 - SQL preprocessor utility (iqsqlpp) 411
- w option
 - SQL preprocessor utility (iqsqlpp) 411
- x option
 - SQL preprocessor utility (iqsqlpp) 411
- z option
 - SQL preprocessor utility (iqsqlpp) 411
- .NET
 - data control 203
 - using theSAP Sybase IQ .NET Data Provider 165
 - .NET API 211
 - about 165
 - .NET Data Provider
 - about 165
 - accessing data 170
 - adding a reference 167
 - connecting to a database 168
 - connection pooling 169
 - dbdata.dll 195
 - deleting data 170
 - deploying 194
 - Entity Framework support 188
 - error handling 187
 - exception handling 187
 - executing stored procedures 185
 - features 166
 - files required for deployment 194
 - iAnywhere.Data.SQLAnywhere provider 166
 - inserting data 170
 - obtaining time values 184
 - POOLING option 169
 - referencing the provider classes in your source code 167
 - registering 195
 - running the sample projects 167
 - supported languages 165
 - system requirements 194
 - tracing support 196
 - transaction processing 186
 - updating data 170
 - using the Simple code sample 200
 - using the Table Viewer code sample 201
 - versions supported 165
 - .NET database programming interfaces
 - tutorial 203
 - [database tools API] Autotune() enumeration 748
 - [database tools API] Checkpoint() enumeration 748
 - [database tools API] History() enumeration 748
 - [database tools API] Padding() enumeration 749
 - [database tools API] Unit() enumeration 749

Index

- [database tools API] Unload() enumeration 749
- [database tools API] UserList() enumeration 749
- [database tools API] Validation() enumeration 750
- [database tools API] Verbosity() enumeration 750
- [database tools API] Version() enumeration 750
- [SQL Anywhere .NET API] SABulkCopyOptions() enumeration 212
- [SQL Anywhere .NET API] SAIsolationLevel() enumeration 213
- [SQL Anywhere C API] a_sqlany_data_direction() enumeration 518
- [SQL Anywhere C API] a_sqlany_data_type() enumeration 518
- [SQL Anywhere C API] a_sqlany_native_type() enumeration 519
- [SQL Anywhere C API] SACAPI_ERROR_SIZE variable 520
- [SQL Anywhere C API] SQLANY_API_VERSION_1 variable 520
- [SQL Anywhere C API] SQLANY_API_VERSION_2 variable 520
- @HttpMethod
 - accessing HTTP headers 641
- @HttpRequestString
 - accessing HTTP headers 641
- @HttpStatus
 - accessing HTTP headers 641
- @HttpURI
 - accessing HTTP headers 641
- @HttpVersion
 - accessing HTTP headers 641
- A**
- a_backup_db structure [database tools API]
 - auto_tune_writers char 751
- a_backup_db structure [database tools API]
 - backup_comment const char * 751
- a_backup_db structure [database tools API]
 - backup_database a_bit_field 751
- a_backup_db structure [database tools API]
 - backup_history char 751
- a_backup_db structure [database tools API]
 - backup_interrupted char 752
- a_backup_db structure [database tools API]
 - backup_logfile a_bit_field 752
- a_backup_db structure [database tools API]
 - chkpt_log_type char 752
- a_backup_db structure [database tools API]
 - confirmrtn MSG_CALLBACK 752
- a_backup_db structure [database tools API]
 - connectparms const char * 752
- a_backup_db structure [database tools API]
 - description 751
- a_backup_db structure [database tools API]
 - errorrtn MSG_CALLBACK 753
- a_backup_db structure [database tools API]
 - hotlog_filename const char * 753
- a_backup_db structure [database tools API]
 - msgsrtn MSG_CALLBACK 753
- a_backup_db structure [database tools API]
 - no_confirm a_bit_field 753
- a_backup_db structure [database tools API]
 - output_dir const char * 753
- a_backup_db structure [database tools API]
 - page_blocksize a_sql_uint32 754
- a_backup_db structure [database tools API]
 - progress_messages a_bit_field 754
- a_backup_db structure [database tools API]
 - quiet a_bit_field 754
- a_backup_db structure [database tools API]
 - rename_local_log a_bit_field 754
- a_backup_db structure [database tools API]
 - rename_log a_bit_field 754
- a_backup_db structure [database tools API]
 - server_backup a_bit_field 755
- a_backup_db structure [database tools API]
 - statusrtn MSG_CALLBACK 755
- a_backup_db structure [database tools API]
 - truncate_log a_bit_field 755
- a_backup_db structure [database tools API]
 - version unsigned short 755
- a_backup_db structure [database tools API]
 - wait_after_end a_bit_field 755
- a_backup_db structure [database tools API]
 - wait_before_start a_bit_field 756
- a_change_log structure [database tools API]
 - change_logname a_bit_field 756
- a_change_log structure [database tools API]
 - change_mirrorname a_bit_field 756
- a_change_log structure [database tools API]
 - dbname const char * 756
- a_change_log structure [database tools API]
 - description 756
- a_change_log structure [database tools API]
 - encryption_key char * 757

- a_change_log structure [database tools API]
 - errortn MSG_CALLBACK 757
- a_change_log structure [database tools API]
 - generation_number unsigned short 757
- a_change_log structure [database tools API]
 - ignore_dbsync_trunc a_bit_field 757
- a_change_log structure [database tools API]
 - ignore_ltm_trunc a_bit_field 757
- a_change_log structure [database tools API]
 - ignore_remote_trunc a_bit_field 757
- a_change_log structure [database tools API]
 - logname const char * 758
- a_change_log structure [database tools API]
 - mirrorname const char * 758
- a_change_log structure [database tools API] msgtrn
 - MSG_CALLBACK 758
- a_change_log structure [database tools API]
 - query_only a_bit_field 758
- a_change_log structure [database tools API] quiet
 - a_bit_field 758
- a_change_log structure [database tools API]
 - set_generation_number a_bit_field 758
- a_change_log structure [database tools API] version
 - unsigned short 759
- a_change_log structure [database tools API]
 - zap_current_offset char * 759
- a_change_log structure [database tools API]
 - zap_starting_offset char * 759
- a_create_db structure [database tools API]
 - accent_sensitivity char 759
- a_create_db structure [database tools API]
 - avoid_view_collisions a_bit_field 760
- a_create_db structure [database tools API]
 - blank_pad a_bit_field 760
- a_create_db structure [database tools API]
 - case_sensitivity_use_default a_bit_field 760
- a_create_db structure [database tools API]
 - checksum a_bit_field 760
- a_create_db structure [database tools API]
 - data_store_type const char * 761
- a_create_db structure [database tools API] db_size
 - unsigned int 761
- a_create_db structure [database tools API]
 - db_size_unit int 761
- a_create_db structure [database tools API] dba_pwd
 - char * 761
- a_create_db structure [database tools API] dba_uid
 - char * 761
- a_create_db structure [database tools API] dbname
 - const char * 761
- a_create_db structure [database tools API]
 - default_collation const char * 762
- a_create_db structure [database tools API]
 - description 759
- a_create_db structure [database tools API] encoding
 - const char * 762
- a_create_db structure [database tools API] encrypt
 - a_bit_field 762
- a_create_db structure [database tools API]
 - encrypted_tables a_bit_field 762
- a_create_db structure [database tools API]
 - encryption_algorithm const char * 762
- a_create_db structure [database tools API]
 - encryption_key const char * 763
- a_create_db structure [database tools API] errortn
 - MSG_CALLBACK 763
- a_create_db structure [database tools API]
 - iq_params void * 763
- a_create_db structure [database tools API] jconnect
 - a_bit_field 763
- a_create_db structure [database tools API] logname
 - const char * 763
- a_create_db structure [database tools API]
 - mirrorname const char * 763
- a_create_db structure [database tools API] msgtrn
 - MSG_CALLBACK 764
- a_create_db structure [database tools API]
 - nchar_collation const char * 764
- a_create_db structure [database tools API]
 - page_size unsigned short 764
- a_create_db structure [database tools API]
 - respect_case a_bit_field 764
- a_create_db structure [database tools API] startline
 - const char * 764
- a_create_db structure [database tools API]
 - sys_proc_definer a_bit_field 765
- a_create_db structure [database tools API] verbose
 - char 765
- a_create_db structure [database tools API] version
 - unsigned short 765
- a_db_info structure [database tools API]
 - bit_map_pages a_sql_uint32 765
- a_db_info structure [database tools API]
 - charcollationspecbuffer char * 765
- a_db_info structure [database tools API]
 - charcollationspecbufsize unsigned short 766

Index

- a_db_info structure [database tools API]
 - charencodingbuffer char * 766
- a_db_info structure [database tools API]
 - charencodingbufsize unsigned short 766
- a_db_info structure [database tools API] checksum
 - a_bit_field 766
- a_db_info structure [database tools API]
 - connectparms const char * 766
- a_db_info structure [database tools API] dbbufsize
 - unsigned short 767
- a_db_info structure [database tools API]
 - dbnamebuffer char * 767
- a_db_info structure [database tools API] description
 - 765
- a_db_info structure [database tools API]
 - encrypted_tables a_bit_field 767
- a_db_info structure [database tools API] errortrn
 - MSG_CALLBACK 767
- a_db_info structure [database tools API] file_size
 - a_sql_uint32 767
- a_db_info structure [database tools API] free_pages
 - a_sql_uint32 767
- a_db_info structure [database tools API] logbufsize
 - unsigned short 767
- a_db_info structure [database tools API]
 - lognamebuffer char * 768
- a_db_info structure [database tools API]
 - mirrorbufsize unsigned short 768
- a_db_info structure [database tools API]
 - mirrornamebuffer char * 768
- a_db_info structure [database tools API] msgtrn
 - MSG_CALLBACK 768
- a_db_info structure [database tools API]
 - ncharcollationspecbuffer char * 768
- a_db_info structure [database tools API]
 - ncharcollationspecbufsize unsigned short 768
- a_db_info structure [database tools API]
 - ncharencodingbuffer char * 768
- a_db_info structure [database tools API]
 - ncharencodingbufsize unsigned short 769
- a_db_info structure [database tools API]
 - other_pages a_sql_uint32 769
- a_db_info structure [database tools API]
 - page_usage a_bit_field 769
- a_db_info structure [database tools API] quiet
 - a_bit_field 769
- a_db_info structure [database tools API] statusrtn
 - MSG_CALLBACK 769
- a_db_info structure [database tools API] sysinfo
 - a_sysinfo 769
- a_db_info structure [database tools API] totals
 - a_table_info * 770
- a_db_info structure [database tools API] version
 - unsigned short 770
- a_db_version_info structure [database tools API]
 - created_version char 770
- a_db_version_info structure [database tools API]
 - description 770
- a_db_version_info structure [database tools API]
 - errortrn MSG_CALLBACK 770
- a_db_version_info structure [database tools API]
 - filename const char * 770
- a_db_version_info structure [database tools API]
 - msgtrn MSG_CALLBACK 771
- a_db_version_info structure [database tools API]
 - version unsigned short 771
- a_dblic_info structure [database tools API]
 - compname char * 771
- a_dblic_info structure [database tools API]
 - conncount a_sql_int32 771
- a_dblic_info structure [database tools API]
 - description 771
- a_dblic_info structure [database tools API] errortrn
 - MSG_CALLBACK 771
- a_dblic_info structure [database tools API]
 - exename char * 772
- a_dblic_info structure [database tools API]
 - installkey char * 772
- a_dblic_info structure [database tools API] msgtrn
 - MSG_CALLBACK 772
- a_dblic_info structure [database tools API]
 - nodecount a_sql_int32 772
- a_dblic_info structure [database tools API]
 - query_only a_bit_field 772
- a_dblic_info structure [database tools API] quiet
 - a_bit_field 772
- a_dblic_info structure [database tools API] type
 - a_license_type 773
- a_dblic_info structure [database tools API]
 - username char * 773
- a_dblic_info structure [database tools API] version
 - unsigned short 773
- a_dbtools_info structure [database tools API]
 - description 773

- a_dbtools_info structure [database tools API]
 - errortn MSG_CALLBACK 773
- a_log_file_info structure [database tools API]
 - dbname const char * 774
- a_log_file_info structure [database tools API]
 - description 773
- a_log_file_info structure [database tools API]
 - encryption_key const char * 774
- a_log_file_info structure [database tools API]
 - errortn MSG_CALLBACK 774
- a_log_file_info structure [database tools API]
 - logname char * 774
- a_log_file_info structure [database tools API]
 - logname_size size_t 774
- a_log_file_info structure [database tools API]
 - mirrorname char * 774
- a_log_file_info structure [database tools API]
 - mirrorname_size size_t 774
- a_log_file_info structure [database tools API]
 - reserved void * 775
- a_log_file_info structure [database tools API]
 - version unsigned short 775
- a_name structure [database tools API]
 - description 775
- a_name structure [database tools API]
 - name char 775
- a_name structure [database tools API]
 - next struct a_name * 775
- a_remote_sql structure [database tools API]
 - apply a_bit_field 776
- a_remote_sql structure [database tools API]
 - argv char ** 776
- a_remote_sql structure [database tools API]
 - batch a_bit_field 777
- a_remote_sql structure [database tools API]
 - confirmrtn MSG_CALLBACK 777
- a_remote_sql structure [database tools API]
 - connectparms char * 777
- a_remote_sql structure [database tools API]
 - debug a_bit_field 777
- a_remote_sql structure [database tools API]
 - debug_dump_size a_sql_uint32 778
- a_remote_sql structure [database tools API]
 - debug_page_offsets a_bit_field 778
- a_remote_sql structure [database tools API]
 - default_window_title char * 778
- a_remote_sql structure [database tools API]
 - deleted a_bit_field 778
- a_remote_sql structure [database tools API]
 - description 775
- a_remote_sql structure [database tools API]
 - encryption_key char * 778
- a_remote_sql structure [database tools API]
 - errortn MSG_CALLBACK 778
- a_remote_sql structure [database tools API]
 - frequency a_sql_uint32 779
- a_remote_sql structure [database tools API]
 - full_q_scan a_bit_field 779
- a_remote_sql structure [database tools API]
 - include_scan_range char * 779
- a_remote_sql structure [database tools API]
 - latest_backup a_bit_field 779
- a_remote_sql structure [database tools API]
 - link_debug a_bit_field 779
- a_remote_sql structure [database tools API]
 - locale char * 779
- a_remote_sql structure [database tools API]
 - log_file_name const char * 780
- a_remote_sql structure [database tools API]
 - log_size a_sql_uint32 780
- a_remote_sql structure [database tools API]
 - logrtn MSG_CALLBACK 780
- a_remote_sql structure [database tools API]
 - max_length a_sql_uint32 780
- a_remote_sql structure [database tools API]
 - memory a_sql_uint32 781
- a_remote_sql structure [database tools API]
 - mirror_logs char * 781
- a_remote_sql structure [database tools API]
 - more a_bit_field 781
- a_remote_sql structure [database tools API]
 - msgqueue rtn MSG_QUEUE_CALLBACK 781
- a_remote_sql structure [database tools API]
 - msg rtn MSG_CALLBACK 781
- a_remote_sql structure [database tools API]
 - no_user_interaction a_bit_field 782
- a_remote_sql structure [database tools API]
 - operations a_sql_uint32 782
- a_remote_sql structure [database tools API]
 - patience_retry a_sql_uint32 782
- a_remote_sql structure [database tools API]
 - progress_index rtn SET_PROGRESS_CALLBACK 782
- a_remote_sql structure [database tools API]
 - progress_msg rtn MSG_CALLBACK 783

Index

- a_remote_sql structure [database tools API] queueparms char * 783
- a_remote_sql structure [database tools API] receive a_bit_field 783
- a_remote_sql structure [database tools API] receive_delay a_sql_uint32 783
- a_remote_sql structure [database tools API] remote_output_file_name char * 783
- a_remote_sql structure [database tools API] rename_log a_bit_field 784
- a_remote_sql structure [database tools API] resend_urgency a_sql_uint32 784
- a_remote_sql structure [database tools API] scan_log a_bit_field 784
- a_remote_sql structure [database tools API] send a_bit_field 784
- a_remote_sql structure [database tools API] send_delay a_sql_uint32 784
- a_remote_sql structure [database tools API] set_window_title_rtn SET_WINDOW_TITLE_CALLBACK 785
- a_remote_sql structure [database tools API] threads a_sql_uint32 785
- a_remote_sql structure [database tools API] transaction_logs char * 785
- a_remote_sql structure [database tools API] triggers a_bit_field 785
- a_remote_sql structure [database tools API] truncate_remote_output_file a_bit_field 785
- a_remote_sql structure [database tools API] unused a_bit_field 786
- a_remote_sql structure [database tools API] use_hex_offsets a_bit_field 786
- a_remote_sql structure [database tools API] use_relative_offsets a_bit_field 786
- a_remote_sql structure [database tools API] verbose a_bit_field 786
- a_remote_sql structure [database tools API] version unsigned short 786
- a_remote_sql structure [database tools API] warningrtn MSG_CALLBACK 787
- a_sqlany_bind_param structure [SQL Anywhere C API] description 528
- a_sqlany_bind_param structure [SQL Anywhere C API] direction a_sqlany_data_direction 528
- a_sqlany_bind_param structure [SQL Anywhere C API] name char * 528
- a_sqlany_bind_param structure [SQL Anywhere C API] value a_sqlany_data_value 528
- a_sqlany_bind_param_info structure [SQL Anywhere C API] description 528
- a_sqlany_bind_param_info structure [SQL Anywhere C API] direction a_sqlany_data_direction 529
- a_sqlany_bind_param_info structure [SQL Anywhere C API] input_value a_sqlany_data_value 529
- a_sqlany_bind_param_info structure [SQL Anywhere C API] name char * 529
- a_sqlany_bind_param_info structure [SQL Anywhere C API] output_value a_sqlany_data_value 529
- a_sqlany_column_info structure [SQL Anywhere C API] description 529
- a_sqlany_column_info structure [SQL Anywhere C API] max_size size_t 530
- a_sqlany_column_info structure [SQL Anywhere C API] name char * 530
- a_sqlany_column_info structure [SQL Anywhere C API] native_type a_sqlany_native_type 530
- a_sqlany_column_info structure [SQL Anywhere C API] nullable sacapi_bool 530
- a_sqlany_column_info structure [SQL Anywhere C API] precision unsigned short 530
- a_sqlany_column_info structure [SQL Anywhere C API] scale unsigned short 531
- a_sqlany_column_info structure [SQL Anywhere C API] type a_sqlany_data_type 531
- a_sqlany_data_direction() enumeration [SQL Anywhere C API] 518
- a_sqlany_data_info structure [SQL Anywhere C API] data_size size_t 531
- a_sqlany_data_info structure [SQL Anywhere C API] description 531
- a_sqlany_data_info structure [SQL Anywhere C API] is_null sacapi_bool 531
- a_sqlany_data_info structure [SQL Anywhere C API] type a_sqlany_data_type 532
- a_sqlany_data_type() enumeration [SQL Anywhere C API] 518
- a_sqlany_data_value structure [SQL Anywhere C API] buffer char * 532

- a_sqlany_data_value structure [SQL Anywhere C API] buffer_size size_t 532
- a_sqlany_data_value structure [SQL Anywhere C API] description 532
- a_sqlany_data_value structure [SQL Anywhere C API] is_null sacapi_bool * 533
- a_sqlany_data_value structure [SQL Anywhere C API] length size_t * 533
- a_sqlany_data_value structure [SQL Anywhere C API] type a_sqlany_data_type 533
- a_sqlany_native_type() enumeration [SQL Anywhere C API] 519
- a_sync_db structure [database tools API] allow_outside_connect a_bit_field 787
- a_sync_db structure [database tools API] allow_schema_change a_bit_field 787
- a_sync_db structure [database tools API] apply_dnld_file const char * 787
- a_sync_db structure [database tools API] argv char ** 788
- a_sync_db structure [database tools API] autoclose a_bit_field 788
- a_sync_db structure [database tools API] background_retry a_sql_uint32 788
- a_sync_db structure [database tools API] background_sync a_bit_field 788
- a_sync_db structure [database tools API] cache_verbosity a_bit_field 788
- a_sync_db structure [database tools API] ce_argv char ** 789
- a_sync_db structure [database tools API] ce_reproc_argv char ** 789
- a_sync_db structure [database tools API] changing_pwd a_bit_field 789
- a_sync_db structure [database tools API] confirmrtn MSG_CALLBACK 789
- a_sync_db structure [database tools API] connectparms char * 789
- a_sync_db structure [database tools API] connectparms_allocated a_bit_field 790
- a_sync_db structure [database tools API] continue_download a_bit_field 790
- a_sync_db structure [database tools API] create_dnld_file const char * 790
- a_sync_db structure [database tools API] debug a_bit_field 790
- a_sync_db structure [database tools API] debug_dump_char a_bit_field 790
- a_sync_db structure [database tools API] debug_dump_hex a_bit_field 790
- a_sync_db structure [database tools API] debug_dump_size a_sql_uint32 791
- a_sync_db structure [database tools API] debug_page_offsets a_bit_field 791
- a_sync_db structure [database tools API] default_window_title char * 791
- a_sync_db structure [database tools API] description 787
- a_sync_db structure [database tools API] dl_insert_width a_sql_uint32 791
- a_sync_db structure [database tools API] dl_use_put a_bit_field 791
- a_sync_db structure [database tools API] dlg_info_msg a_sql_uint32 791
- a_sync_db structure [database tools API] dnld_fail_len a_sql_uint32 791
- a_sync_db structure [database tools API] dnld_file_extra const char * 792
- a_sync_db structure [database tools API] dnld_gen_num a_bit_field 792
- a_sync_db structure [database tools API] dnld_read_size a_sql_uint32 792
- a_sync_db structure [database tools API] download_only a_bit_field 792
- a_sync_db structure [database tools API] encrypted_stream_opts const char * 792
- a_sync_db structure [database tools API] encryption_key char * 793
- a_sync_db structure [database tools API] entered_dialog a_bit_field 793
- a_sync_db structure [database tools API] errorrtn MSG_CALLBACK 793
- a_sync_db structure [database tools API] est_upld_row_cnt a_sql_uint32 793
- a_sync_db structure [database tools API] extended_options char * 793
- a_sync_db structure [database tools API] hide_conn_str a_bit_field 794
- a_sync_db structure [database tools API] hide_ml_pwd a_bit_field 794
- a_sync_db structure [database tools API] hovering_frequency a_sql_uint32 794
- a_sync_db structure [database tools API] ignore_debug_interrupt a_bit_field 794
- a_sync_db structure [database tools API] ignore_hook_errors a_bit_field 794

Index

- a_sync_db structure [database tools API]
 - ignore_hovering a_bit_field 795
- a_sync_db structure [database tools API]
 - ignore_scheduling a_bit_field 795
- a_sync_db structure [database tools API]
 - include_scan_range const char * 795
- a_sync_db structure [database tools API] init_cache
 - a_sql_uint32 795
- a_sync_db structure [database tools API]
 - init_cache_suffix char 795
- a_sync_db structure [database tools API]
 - kill_other_connections a_bit_field 796
- a_sync_db structure [database tools API]
 - last_upload_def a_syncpub * 796
- a_sync_db structure [database tools API]
 - lite_blob_handling a_bit_field 796
- a_sync_db structure [database tools API]
 - log_file_name const char * 796
- a_sync_db structure [database tools API] log_size
 - a_sql_uint32 796
- a_sync_db structure [database tools API] logrtn
 - MSG_CALLBACK 797
- a_sync_db structure [database tools API]
 - max_cache a_sql_uint32 797
- a_sync_db structure [database tools API]
 - max_cache_suffix char 797
- a_sync_db structure [database tools API]
 - min_cache a_sql_uint32 797
- a_sync_db structure [database tools API]
 - min_cache_suffix char 797
- a_sync_db structure [database tools API]
 - mlpassword char * 797
- a_sync_db structure [database tools API]
 - msgqueuerrtn
 - MSG_QUEUE_CALLBACK 798
- a_sync_db structure [database tools API] msggrtn
 - MSG_CALLBACK 798
- a_sync_db structure [database tools API]
 - new_mlpassword char * 798
- a_sync_db structure [database tools API]
 - no_offline_logscan a_sql_uint32 798
- a_sync_db structure [database tools API]
 - no_schema_cache a_bit_field 799
- a_sync_db structure [database tools API]
 - no_stream_compress a_bit_field 799
- a_sync_db structure [database tools API] offline_dir
 - const char * 799
- a_sync_db structure [database tools API]
 - output_to_file a_bit_field 799
- a_sync_db structure [database tools API]
 - output_to_mobile_link a_bit_field 799
- a_sync_db structure [database tools API]
 - persist_connection a_bit_field 799
- a_sync_db structure [database tools API] ping
 - a_bit_field 800
- a_sync_db structure [database tools API]
 - preload_dlls char * 800
- a_sync_db structure [database tools API]
 - progress_index_rtn
 - SET_PROGRESS_CALLBACK 800
- a_sync_db structure [database tools API]
 - progress_msg_rtn MSG_CALLBACK 800
- a_sync_db structure [database tools API]
 - prompt_again a_bit_field 800
- a_sync_db structure [database tools API]
 - prompt_for_encrypt_key a_bit_field 800
- a_sync_db structure [database tools API]
 - protocol_add_cli_bit_to_cli_both a_bit_field 801
- a_sync_db structure [database tools API]
 - protocol_add_cli_bit_to_cli_max a_bit_field 801
- a_sync_db structure [database tools API]
 - protocol_add_serv_bit_to_cli_both a_bit_field 801
- a_sync_db structure [database tools API]
 - protocol_add_serv_bit_to_cli_max a_bit_field 801
- a_sync_db structure [database tools API]
 - protocol_add_serv_bit_to_serv_both a_bit_field 801
- a_sync_db structure [database tools API]
 - protocol_add_serv_bit_to_serv_max a_bit_field 801
- a_sync_db structure [database tools API] raw_file
 - const char * 801
- a_sync_db structure [database tools API]
 - rename_log a_bit_field 802
- a_sync_db structure [database tools API] reserved
 - a_bit_field 802
- a_sync_db structure [database tools API]
 - retry_remote_ahead a_bit_field 802
- a_sync_db structure [database tools API]
 - retry_remote_behind a_bit_field 802
- a_sync_db structure [database tools API]
 - server_mode a_bit_field 802

- a_sync_db structure [database tools API]
 - server_port a_sql_uint32 803
- a_sync_db structure [database tools API]
 - set_window_title_rtn
 - SET_WINDOW_TITLE_CALLBACK
 - 803
- a_sync_db structure [database tools API] status_rtn
 - STATUS_CALLBACK 803
- a_sync_db structure [database tools API]
 - strictly_free_memory a_bit_field 803
- a_sync_db structure [database tools API]
 - strictly_ignore_trigger_ops a_bit_field
 - 803
- a_sync_db structure [database tools API] sync_opt
 - char * 803
- a_sync_db structure [database tools API]
 - sync_params char * 804
- a_sync_db structure [database tools API]
 - sync_profile char * 804
- a_sync_db structure [database tools API]
 - trans_upload a_bit_field 804
- a_sync_db structure [database tools API]
 - upld_fail_len a_sql_uint32 804
- a_sync_db structure [database tools API]
 - upload_defs a_syncpub * 804
- a_sync_db structure [database tools API]
 - upload_only a_bit_field 805
- a_sync_db structure [database tools API] usage_rtn
 - USAGE_CALLBACK 805
- a_sync_db structure [database tools API]
 - use_fixed_cache a_bit_field 805
- a_sync_db structure [database tools API]
 - use_hex_offsets a_bit_field 805
- a_sync_db structure [database tools API]
 - use_relative_offsets a_bit_field 805
- a_sync_db structure [database tools API]
 - used_dialog_allocation a_bit_field 805
- a_sync_db structure [database tools API]
 - user_name char * 806
- a_sync_db structure [database tools API] verbose
 - a_bit_field 806
- a_sync_db structure [database tools API]
 - verbose_download a_bit_field 806
- a_sync_db structure [database tools API]
 - verbose_download_data a_bit_field 806
- a_sync_db structure [database tools API]
 - verbose_hook a_bit_field 806
- a_sync_db structure [database tools API]
 - verbose_minimum a_bit_field 806
- a_sync_db structure [database tools API]
 - verbose_msgid a_bit_field 807
- a_sync_db structure [database tools API]
 - verbose_option_info a_bit_field 807
- a_sync_db structure [database tools API]
 - verbose_protocol a_bit_field 807
- a_sync_db structure [database tools API]
 - verbose_row_cnts a_bit_field 807
- a_sync_db structure [database tools API]
 - verbose_row_data a_bit_field 807
- a_sync_db structure [database tools API]
 - verbose_server a_bit_field 808
- a_sync_db structure [database tools API]
 - verbose_upload a_bit_field 808
- a_sync_db structure [database tools API]
 - verbose_upload_data a_bit_field 808
- a_sync_db structure [database tools API] version
 - unsigned short 808
- a_sync_db structure [database tools API]
 - warning_rtn MSG_CALLBACK 808
- a_syncpub structure [database tools API]
 - description 808
- a_syncpub structure [database tools API] ext_opt
 - char * 809
- a_syncpub structure [database tools API] next_struct
 - a_syncpub * 809
- a_syncpub structure [database tools API] pub_name
 - char * 809
- a_syncpub structure [database tools API]
 - subscription char * 809
- a_sysinfo structure [database tools API]
 - blank_padding a_bit_field 810
- a_sysinfo structure [database tools API]
 - case_sensitivity a_bit_field 810
- a_sysinfo structure [database tools API]
 - default_collation char 810
- a_sysinfo structure [database tools API] description
 - 809
- a_sysinfo structure [database tools API] encryption
 - a_bit_field 810
- a_sysinfo structure [database tools API] page_size
 - unsigned short 810
- a_sysinfo structure [database tools API] valid_data
 - a_bit_field 810
- a_table_info structure [database tools API]
 - description 810
- a_table_info structure [database tools API]
 - index_pages a_sql_uint32 811

Index

- a_table_info structure [database tools API]
 - index_used a_sql_uint32 811
- a_table_info structure [database tools API]
 - index_used_pct a_sql_uint32 811
- a_table_info structure [database tools API] next
 - struct a_table_info * 811
- a_table_info structure [database tools API] table_id
 - a_sql_uint32 811
- a_table_info structure [database tools API]
 - table_name char * 811
- a_table_info structure [database tools API]
 - table_pages a_sql_uint32 811
- a_table_info structure [database tools API]
 - table_used a_sql_uint32 812
- a_table_info structure [database tools API]
 - table_used_pct a_sql_uint32 812
- a_translate_log structure [database tools API]
 - ansi_sql a_bit_field 812
- a_translate_log structure [database tools API]
 - chronological_order a_bit_field 812
- a_translate_log structure [database tools API]
 - comment_trigger_trans a_bit_field 812
- a_translate_log structure [database tools API]
 - confirmrtn MSG_CALLBACK 813
- a_translate_log structure [database tools API]
 - connectparms const char * 813
- a_translate_log structure [database tools API] debug
 - a_bit_field 813
- a_translate_log structure [database tools API]
 - debug_dump_char a_bit_field 813
- a_translate_log structure [database tools API]
 - debug_dump_hex a_bit_field 813
- a_translate_log structure [database tools API]
 - debug_dump_size a_sql_uint32 814
- a_translate_log structure [database tools API]
 - debug_page_offsets a_bit_field 814
- a_translate_log structure [database tools API]
 - debug_sql_remote a_bit_field 814
- a_translate_log structure [database tools API]
 - description 812
- a_translate_log structure [database tools API]
 - encryption_key const char * 814
- a_translate_log structure [database tools API]
 - errortn MSG_CALLBACK 814
- a_translate_log structure [database tools API]
 - extra_audit a_bit_field 814
- a_translate_log structure [database tools API]
 - force_chaining a_bit_field 814
- a_translate_log structure [database tools API]
 - force_recovery a_bit_field 815
- a_translate_log structure [database tools API]
 - generate_reciprocals a_bit_field 815
- a_translate_log structure [database tools API]
 - include_audit a_bit_field 815
- a_translate_log structure [database tools API]
 - include_destination_sets const char * 815
- a_translate_log structure [database tools API]
 - include_publications const char * 815
- a_translate_log structure [database tools API]
 - include_scan_range const char * 815
- a_translate_log structure [database tools API]
 - include_source_sets const char * 815
- a_translate_log structure [database tools API]
 - include_subsets a_bit_field 816
- a_translate_log structure [database tools API]
 - include_tables const char * 816
- a_translate_log structure [database tools API]
 - include_trigger_trans a_bit_field 816
- a_translate_log structure [database tools API]
 - leave_output_on_error a_bit_field 816
- a_translate_log structure [database tools API]
 - logname const char * 816
- a_translate_log structure [database tools API] logrtn
 - MSG_CALLBACK 816
- a_translate_log structure [database tools API]
 - logs_dir const char * 817
- a_translate_log structure [database tools API]
 - match_mode a_bit_field 817
- a_translate_log structure [database tools API]
 - match_pos const char * 817
- a_translate_log structure [database tools API]
 - msgtrtn MSG_CALLBACK 817
- a_translate_log structure [database tools API]
 - omit_comments a_bit_field 817
- a_translate_log structure [database tools API]
 - queparms const char * 817
- a_translate_log structure [database tools API] quiet
 - a_bit_field 818
- a_translate_log structure [database tools API]
 - recovery_bytes a_sql_uint32 818
- a_translate_log structure [database tools API]
 - recovery_ops a_sql_uint32 818
- a_translate_log structure [database tools API]
 - remove_rollback a_bit_field 818
- a_translate_log structure [database tools API]
 - replace a_bit_field 818

- a_translate_log structure [database tools API]
 - repserver_users const char * 819
- a_translate_log structure [database tools API]
 - show_undo a_bit_field 819
- a_translate_log structure [database tools API]
 - since_checkpoint a_bit_field 819
- a_translate_log structure [database tools API]
 - since_time a_sql_uint32 819
- a_translate_log structure [database tools API]
 - sqlname const char * 819
- a_translate_log structure [database tools API]
 - statusrtn MSG_CALLBACK 820
- a_translate_log structure [database tools API]
 - use_hex_offsets a_bit_field 820
- a_translate_log structure [database tools API]
 - use_relative_offsets a_bit_field 820
- a_translate_log structure [database tools API]
 - userlist p_name 820
- a_translate_log structure [database tools API]
 - userlisttype char 820
- a_translate_log structure [database tools API]
 - version unsigned short 820
- a_truncate_log structure [database tools API]
 - connectparms const char * 821
- a_truncate_log structure [database tools API]
 - description 821
- a_truncate_log structure [database tools API]
 - errorrtn MSG_CALLBACK 821
- a_truncate_log structure [database tools API]
 - msgtrtn MSG_CALLBACK 821
- a_truncate_log structure [database tools API] quiet
 - a_bit_field 821
- a_truncate_log structure [database tools API]
 - server_backup a_bit_field 822
- a_truncate_log structure [database tools API]
 - truncate_interrupted char 822
- a_truncate_log structure [database tools API]
 - version unsigned short 822
- a_v4_extfn_blob
 - blob 17
 - blob_length 18
 - close_istream 19
 - open_istream 19
 - release 20
 - structure 17
- a_v4_extfn_blob_istream
 - blob input stream 21
 - get 21
 - structure 21
- a_v4_extfn_col_subset_of_input
 - column values subset 25
 - structure 25
- a_v4_extfn_column_data
 - column data 22
 - structure 22
- a_v4_extfn_column_list
 - column list 23
 - structure 23
- a_v4_extfn_describe_col_type enumerator 90
- a_v4_extfn_describe_parm_type enumerator 91
- a_v4_extfn_describe_return enumerator 93
- a_v4_extfn_describe_udf_type enumerator 95
- a_v4_extfn_estimate
 - optimizer estimate 113
 - structure 113
- a_v4_extfn_license_info 112
- a_v4_extfn_order_el
 - column order 24
 - structure 24
- a_v4_extfn_orderby_list
 - order by list 113
 - structure 113
- a_v4_extfn_partitionby_col_num enumerator 114
- a_v4_extfn_proc
 - external function 97
 - structure 97
- a_v4_extfn_proc_context
 - convert_value method 107
 - external procedure context 100
 - get_blob method 111
 - get_is_cancelled method 105
 - get_value method 102
 - get_value_is_constant method 104
 - log_message method 107
 - set_error method 106
 - set_value method 104
 - structure 100
- a_v4_extfn_row 115
- a_v4_extfn_row_block 116
- a_v4_extfn_state enumerator 95
- a_v4_extfn_table
 - structure 116
 - table 116
- a_v4_extfn_table_context
 - get_blob method 124
 - structure 117
 - table context 117

Index

- a_v4_extfn_table_func
 - structure 124
 - table functions 124
- a_validate_db structure [database tools API]
 - connectparms const char * 822
- a_validate_db structure [database tools API]
 - description 822
- a_validate_db structure [database tools API]
 - errortn MSG_CALLBACK 823
- a_validate_db structure [database tools API] index
 - a_bit_field 823
- a_validate_db structure [database tools API] msg rtn
 - MSG_CALLBACK 823
- a_validate_db structure [database tools API] quiet
 - a_bit_field 823
- a_validate_db structure [database tools API]
 - statusrtn MSG_CALLBACK 823
- a_validate_db structure [database tools API] tables
 - p_name 823
- a_validate_db structure [database tools API] type
 - char 824
- a_validate_db structure [database tools API] version
 - unsigned short 824
- Abort property SARowsCopiedEventArgs class
 - [SQL Anywhere .NET API] 307
- about 423–426, 435, 436, 481, 482
- accent_sensitivity chara_create_db structure
 - [database tools API] 759
- Accept
 - accessing HTTP headers 641
- Accept-Charset
 - accessing HTTP headers 641
- Accept-Encoding
 - accessing HTTP headers 641
- Accept-Language
 - accessing HTTP headers 641
- AcceptCharset option
 - example 652
- ActiveX Data Objects
 - about 320
- Adaptive Server server 946
- Adaptive Server servers 944
- addBatch
 - PreparedStatement class 398
 - Statement class 393
- addShutdownHook
 - Java VM shutdown hooks 374
- administration tools
 - dbtools 731
- ADO
 - about 320
 - Command object 322
 - commands 322
 - Connection object 321
 - connections 321
 - cursor types 142
 - cursors 158
 - introduction to programming 319
 - queries 323
 - Recordset object 323
 - Recordset object and cursor types 324
 - transactions 326
 - updates 325
 - updating data through a cursor 325
 - using SQL statements in applications 131
- ADO.NET
 - about 165
 - autocommit mode 161
 - controlling autocommit behavior 161
 - cursor support 158
 - prepared statements 133
 - using SQL statements in applications 131
- ADO.NET API
 - about 165
- aggregate functions 859
 - statistical 881
 - STDDEV_POP 882
 - STDDEV_SAMP 882
 - VAR_POP 882
 - VAR_SAMP 882
- All property SACCommLinksOptionsBuilder class
 - [SQL Anywhere .NET API] 224
- alloc
 - v4 API method 109
- alloc_sqlda function
 - about 467
- alloc_sqlda_noind function
 - about 468
- ALLOW_NULLS_BY_DEFAULT option
 - Open Client 6
- allow_outside_connect a_bit_fielda_sync_db
 - structure [database tools API] 787
- allow_schema_change a_bit_fielda_sync_db
 - structure [database tools API] 787
- ALTER SERVER statement
 - syntax 961
- altering
 - web services 631

- an_erase_db structure [database tools API]
 - confirmrtn MSG_CALLBACK 824
- an_erase_db structure [database tools API] dbname
 - const char * 824
- an_erase_db structure [database tools API]
 - description 824
- an_erase_db structure [database tools API]
 - encryption_key const char * 824
- an_erase_db structure [database tools API] erase
 - a_bit_field 825
- an_erase_db structure [database tools API] errorrtn
 - MSG_CALLBACK 825
- an_erase_db structure [database tools API] msgrtn
 - MSG_CALLBACK 825
- an_erase_db structure [database tools API] quiet
 - a_bit_field 825
- an_erase_db structure [database tools API] version
 - unsigned short 825
- an_unload_db structure [database tools API]
 - compress_output a_bit_field 826
- an_unload_db structure [database tools API]
 - confirmrtn MSG_CALLBACK 826
- an_unload_db structure [database tools API]
 - connectparms const char * 826
- an_unload_db structure [database tools API] debug
 - a_bit_field 827
- an_unload_db structure [database tools API]
 - description 826
- an_unload_db structure [database tools API]
 - display_create a_bit_field 827
- an_unload_db structure [database tools API]
 - display_create_dbinit a_bit_field 827
- an_unload_db structure [database tools API]
 - encrypted_tables a_bit_field 827
- an_unload_db structure [database tools API]
 - encryption_algorithm const char * 827
- an_unload_db structure [database tools API]
 - encryption_key const char * 828
- an_unload_db structure [database tools API]
 - errorrtn MSG_CALLBACK 828
- an_unload_db structure [database tools API]
 - escape_char char 828
- an_unload_db structure [database tools API]
 - escape_char_present a_bit_field 828
- an_unload_db structure [database tools API]
 - exclude_foreign_keys a_bit_field 828
- an_unload_db structure [database tools API]
 - exclude_hooks a_bit_field 829
- an_unload_db structure [database tools API]
 - exclude_procedures a_bit_field 829
- an_unload_db structure [database tools API]
 - exclude_tables a_bit_field 829
- an_unload_db structure [database tools API]
 - exclude_triggers a_bit_field 829
- an_unload_db structure [database tools API]
 - exclude_views a_bit_field 829
- an_unload_db structure [database tools API] extract
 - a_bit_field 830
- an_unload_db structure [database tools API]
 - genscript a_bit_field 830
- an_unload_db structure [database tools API]
 - include_where_subscribe a_bit_field 830
- an_unload_db structure [database tools API]
 - isolation_level unsigned short 830
- an_unload_db structure [database tools API]
 - isolation_set a_bit_field 830
- an_unload_db structure [database tools API] locale
 - const char * 831
- an_unload_db structure [database tools API]
 - make_auxiliary a_bit_field 831
- an_unload_db structure [database tools API]
 - ms_filename const char * 831
- an_unload_db structure [database tools API]
 - ms_reserve int 831
- an_unload_db structure [database tools API]
 - ms_size int 831
- an_unload_db structure [database tools API] msgrtn
 - MSG_CALLBACK 831
- an_unload_db structure [database tools API]
 - no_confirm a_bit_field 832
- an_unload_db structure [database tools API]
 - no_reload_status a_bit_field 832
- an_unload_db structure [database tools API]
 - notemp_size long 832
- an_unload_db structure [database tools API]
 - preserve_identity_values a_bit_field 832
- an_unload_db structure [database tools API]
 - preserve_ids a_bit_field 832
- an_unload_db structure [database tools API]
 - profiling_uses_single_dbSPACE
 - a_bit_field 833
- an_unload_db structure [database tools API]
 - recompute a_bit_field 833
- an_unload_db structure [database tools API]
 - refresh_mat_view a_bit_field 833
- an_unload_db structure [database tools API]
 - reload_connectparms char * 833

Index

- an_unload_db structure [database tools API]
 - reload_db_filename char * 833
- an_unload_db structure [database tools API]
 - reload_db_logname char * 834
- an_unload_db structure [database tools API]
 - reload_filename const char * 834
- an_unload_db structure [database tools API]
 - reload_page_size unsigned short 834
- an_unload_db structure [database tools API]
 - remote_dir const char * 834
- an_unload_db structure [database tools API]
 - remove_encrypted_tables a_bit_field 834
- an_unload_db structure [database tools API]
 - replace_db a_bit_field 835
- an_unload_db structure [database tools API]
 - runscript a_bit_field 835
- an_unload_db structure [database tools API]
 - schema_reload a_bit_field 835
- an_unload_db structure [database tools API]
 - site_name const char * 835
- an_unload_db structure [database tools API]
 - start_subscriptions a_bit_field 835
- an_unload_db structure [database tools API]
 - startline const char * 836
- an_unload_db structure [database tools API]
 - startline_name a_bit_field 836
- an_unload_db structure [database tools API]
 - startline_old const char * 836
- an_unload_db structure [database tools API]
 - statusrtn MSG_CALLBACK 836
- an_unload_db structure [database tools API]
 - subscriber_username const char * 836
- an_unload_db structure [database tools API]
 - suppress_statistics a_bit_field 836
- an_unload_db structure [database tools API] sysinfo
 - a_sysinfo 837
- an_unload_db structure [database tools API]
 - table_list p_name 837
- an_unload_db structure [database tools API]
 - table_list_provided a_bit_field 837
- an_unload_db structure [database tools API]
 - temp_dir const char * 837
- an_unload_db structure [database tools API]
 - template_name const char * 837
- an_unload_db structure [database tools API]
 - unload_interrupted char 837
- an_unload_db structure [database tools API]
 - unload_type char 838
- an_unload_db structure [database tools API]
 - unordered a_bit_field 838
- an_unload_db structure [database tools API]
 - use_internal_reload a_bit_field 838
- an_unload_db structure [database tools API]
 - use_internal_unload a_bit_field 838
- an_unload_db structure [database tools API]
 - verbose char 838
- an_unload_db structure [database tools API]
 - version unsigned short 839
- an_upgrade_db structure [database tools API]
 - connectparms const char * 839
- an_upgrade_db structure [database tools API]
 - description 839
- an_upgrade_db structure [database tools API]
 - errortn MSG_CALLBACK 839
- an_upgrade_db structure [database tools API]
 - jconnect a_bit_field 839
- an_upgrade_db structure [database tools API]
 - msgtrn MSG_CALLBACK 840
- an_upgrade_db structure [database tools API] quiet
 - a_bit_field 840
- an_upgrade_db structure [database tools API]
 - restart a_bit_field 840
- an_upgrade_db structure [database tools API]
 - statusrtn MSG_CALLBACK 840
- an_upgrade_db structure [database tools API]
 - sys_proc_definer unsigned short 840
- an_upgrade_db structure [database tools API]
 - version unsigned short 841
- analytic functions
 - DENSE_RANK 873
 - PERCENT_RANK 875
 - PERCENTILE_CONT 888
 - PERCENTILE_DISC 890
 - RANK 871
- analytical functions 843
- ansi_sql a_bit_field a_translate_log structure [database tools API] 812
- APIs
 - ADO API 319
 - ADO.NET 165
 - JDBC API 377
 - OLE DB API 319
 - Perl DBD::SQLAnywhere API 535
 - PHP 566
 - Python Database API 545
 - Ruby APIs 591
 - Sybase Open Client API 617

- AppInfo propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 247
 - applications
 - SQL 131
 - apply a_bit_fielda_remote_sql structure [database tools API] 776
 - apply_dnlld_file const char *a_sync_db structure [database tools API] 787
 - argv char **a_remote_sql structure [database tools API] 776
 - argv char **a_sync_db structure [database tools API] 788
 - ARRAY clause
 - using the FETCH statement 456
 - array fetches
 - about 456
 - ESQL 456
 - ascending order 868
 - ASEJDBC class 944
 - asensitive cursors
 - about 150
 - delete example 145
 - introduction 144
 - update example 146
 - AT clause
 - CREATE EXISTING TABLE 964
 - auto_tune_writers chara_backup_db structure [database tools API] 751
 - autoclose a_bit_fielda_sync_db structure [database tools API] 788
 - autocommit
 - controlling 161
 - implementation 162
 - JDBC 390
 - setting for transactions 161
 - AUTOINCREMENT
 - finding most recent row inserted 140
 - AUTOINCREMENT column default 968
 - AutoStart propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 247
 - AutoStop propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 247
 - Autotune() enumeration [database tools API] 748
 - avoid_view_collisions a_bit_fielda_create_db structure [database tools API] 760
- B**
- background processing
 - callback functions 466
 - background_retry a_sql_int32a_sync_db structure [database tools API] 788
 - background_sync a_bit_fielda_sync_db structure [database tools API] 788
 - backup_comment const char *a_backup_db structure [database tools API] 751
 - backup_database a_bit_fielda_backup_db structure [database tools API] 751
 - backup_history chara_backup_db structure [database tools API] 751
 - backup_interrupted chara_backup_db structure [database tools API] 752
 - backup_logfile a_bit_fielda_backup_db structure [database tools API] 752
 - backups
 - DBTools example 736
 - embedded SQL functions 467
 - batch a_bit_fielda_remote_sql structure [database tools API] 777
 - batch inserts
 - JDBC 398
 - BatchSize propertySABulkCopy class [SQL Anywhere .NET API] 214
 - BatchUpdateException
 - JDBC 393
 - BEGIN TRANSACTION statement
 - remote data access 954
 - BIGINT data type
 - embedded SQL 427
 - binary data types
 - embedded SQL 427
 - bind parameters
 - prepared statements 133
 - bind variables
 - about 440
 - BIT data type
 - embedded SQL 427
 - bit fields
 - using 735
 - bit length 894
 - BIT_LENGTH function 894
 - bit_map_pages a_sql_uint32a_db_info structure [database tools API] 765
 - blank padding 424
 - blank padding of DT_NSTRING 424
 - blank padding of DT_STRING 424
 - blank_pad a_bit_fielda_create_db structure [database tools API] 760

Index

- blank_padding a_bit_fielda_sysinfo structure
 - [database tools API] 810
- blob
 - a_v4_extfn_blob 17
- blob input stream
 - a_v4_extfn_blob_istream 21
- BLOBs
 - embedded SQL 460
 - retrieving in embedded SQL 461, 462
 - sending in embedded SQL 462, 463
- block cursors
 - about 139
 - ODBC 143
- bookmarks
 - about 143
- bound parameters
 - prepared statements 133
- Broadcast propertySATcpOptionsBuilder class
 - [SQL Anywhere .NET API] 311
- BroadcastListener propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 311
- buffer char *a_sqlany_data_value structure [SQL Anywhere C API] 532
- buffer_size size_ta_sqlany_data_value structure [SQL Anywhere C API] 532
- Bulk-Library
 - about 617
- BulkCopyTimeout propertySABulkCopy class [SQL Anywhere .NET API] 214
- byte code
 - Java classes 369
- C**
- C API 495
- C programming language
 - data types 427
 - embedded SQL applications 409
- C#
 - support in .NET Data Provider 165
- C++ API 495
- C++ applications
 - dbtools 731
 - embedded SQL 409
- cache_verbosity a_bit_fielda_sync_db structure [database tools API] 788
- CALL statement
 - embedded SQL 463
- callback 482
 - JDBC 401
- callback functions
 - embedded SQL 466
 - registering 480
- callbacks 481, 482
- canceling requests
 - embedded SQL 466
- CanCreateDataSourceEnumerator
 - propertySAFactory class [SQL Anywhere .NET API] 282
- case_sensitivity a_bit_fielda_sysinfo structure [database tools API] 810
- case_sensitivity_use_default
 - a_bit_fielda_create_db structure [database tools API] 760
- ce_argv char **a_sync_db structure [database tools API] 789
- ce_reproc_argv char **a_sync_db structure [database tools API] 789
- CEIL function 895
- CEILING function 895
- certification
 - partner 1
 - platform 3
- chained mode
 - controlling 161
 - implementation 162
 - transactions 161
- chained option
 - JDBC 390
- CHAINED option
 - Open Client 6
- change_logname a_bit_fielda_change_log structure [database tools API] 756
- change_mirrorname a_bit_fielda_change_log structure [database tools API] 756
- changing_pwd a_bit_fielda_sync_db structure [database tools API] 789
- character data
 - character sets in Embedded SQL 430
 - length in Embedded SQL 430
- character sets
 - setting CHAR character set 473
 - setting NCHAR character set 474
 - web services 652
- character strings
 - embedded SQL 411
- charcollationspecbuffer char *a_db_info structure [database tools API] 765

- charcollationspecbufsize unsigned shorta_db_info structure [database tools API] 766
- charencodingbuffer char *a_db_info structure [database tools API] 766
- charencodingbufsize unsigned shorta_db_info structure [database tools API] 766
- Charset propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 247
- CharsetConversion option
 - example 652
- CHECK conditions
 - about 968
- CHECK ON COMMIT clause
 - referential integrity 968
- Checkpoint() enumeration [database tools API] 748
- checksum a_bit_fielda_create_db structure [database tools API] 760
- checksum a_bit_fielda_db_info structure [database tools API] 766
- chkpt_log_type chara_backup_db structure [database tools API] 752
- chronological_order a_bit_fielda_translate_log structure [database tools API] 812
- CIS (Component Integration Services) 5
- Class.forName method
 - loading iAnywhere JDBC 4.0 driver 381
- classes
 - creating 371
 - installing 371
- CLASSPATH environment variable
 - jConnect 382
 - setting 387
- clauses
 - WITH HOLD 138
- clearBatch
 - Statement class 393
- client
 - time change 488
- client connections
 - OLE DB 320
- client files
 - ESQL client API callback function 480
- client side autocommit
 - about 162
- Client-Library
 - Sybase Open Client 617
- CLIENTPORT clause
 - specifying 666
- ClientPort propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 312
- clients
 - web 658
- close method
 - Python 548
- CLOSE statement
 - using cursors in embedded SQL 453
- close_result_set
 - v4 API method 110
- CodeXchange
 - samples 626
- column data
 - a_v4_extfn_column_data 22
- column list
 - a_v4_extfn_column_list 23
- column number
 - partition by 114
- column order
 - a_v4_extfn_order_el 24
- column subset
 - a_v4_extfn_col_subset_of_input 25
- ColumnMappings propertySABulkCopy class [SQL Anywhere .NET API] 215
- columns
 - constraints 968
- Columns fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 284
- Command ADO object
 - ADO 322
- command line utilities
 - SQL preprocessor (iqsqlpp) syntax 411
- Command propertySARowUpdatedEventArgs class [SQL Anywhere .NET API] 305
- Command propertySARowUpdatingEventArgs class [SQL Anywhere .NET API] 306
- commands
 - ADO Command object 322
- CommandText propertySACCommand class [SQL Anywhere .NET API] 234
- CommandTimeout propertySACCommand class [SQL Anywhere .NET API] 235
- CommandType propertySACCommand class [SQL Anywhere .NET API] 235
- CommBufferSize
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 247
- comment_trigger_trans a_bit_fielda_translate_log structure [database tools API] 812

Index

- commenting
 - web services 634
- commit method
 - Python 549
- COMMIT statement
 - cursors 163
 - JDBC 390
 - remote data access 954
- CommitTrans ADO method
 - ADO programming 326
 - updating data 326
- CommLinks propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 248
- compile and link process
 - about 410
- compilers
 - used with sqlpp 415
- compname char *a_dblic_info structure [database tools API] 771
- Component Integration Services 945
- Compress propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 248
- compress_output_a_bit_fieldan_unload_db structure [database tools API] 826
- CompressionThreshold
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 248
- computing deltas between adjacent rows 869
- confirmrtn MSG_CALLBACKa_backup_db structure [database tools API] 752
- confirmrtn MSG_CALLBACKa_remote_sql structure [database tools API] 777
- confirmrtn MSG_CALLBACKa_sync_db structure [database tools API] 789
- confirmrtn MSG_CALLBACKa_translate_log structure [database tools API] 813
- confirmrtn MSG_CALLBACKan_erase_db structure [database tools API] 824
- confirmrtn MSG_CALLBACKan_unload_db structure [database tools API] 826
- conncount a_sql_int32a_dblic_info structure [database tools API] 771
- connect identifier
 - resolving 943
- connect method
 - Python 548
- connecting
 - OLE DB 320
- Connection
 - accessing HTTP headers 641
- Connection ADO object
 - ADO 321
 - ADO programming 326
- connection defaults 390
- connection parameters
 - OLE DB 327
- connection pooling
 - .NET Data Provider 169
 - OLE DB 329
 - web services 637
- connection properties
 - web services 654
- Connection propertySACCommand class [SQL Anywhere .NET API] 236
- Connection propertySATransaction class [SQL Anywhere .NET API] 316
- connection state
 - .NET Data Provider 170
- CONNECTION_PROPERTY function
 - example 651
- ConnectionLifetime
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 248
- ConnectionName
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 249
- ConnectionPool
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 249
- ConnectionReset
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 249
- connections 390
 - ADO Connection object 321
 - connecting to a database using the .NET Data Provider 168
 - functions 486
 - jConnect 384
 - jConnect URL 383
 - JDBC 380
 - JDBC client applications 384
 - JDBC example 384
 - JDBC in the server 388
 - JDBC server-side example 388
 - licensing web applications 646
 - ODBC functions 346
 - ODBC programming 347

- remote 954
- SQL Anywhere 16 JDBC driver URL 381
- ConnectionString
 - propertySACommLinksOptionsBuilder class [SQL Anywhere .NET API] 225
- ConnectionTimeout
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 249
- connectparms char *a_remote_sql structure [database tools API] 777
- connectparms char *a_sync_db structure [database tools API] 789
- connectparms const char *a_backup_db structure [database tools API] 752
- connectparms const char *a_db_info structure [database tools API] 766
- connectparms const char *a_translate_log structure [database tools API] 813
- connectparms const char *a_truncate_log structure [database tools API] 821
- connectparms const char *a_validate_db structure [database tools API] 822
- connectparms const char *an_unload_db structure [database tools API] 826
- connectparms const char *an_upgrade_db structure [database tools API] 839
- connectparms_allocated a_bit_fielda_sync_db structure [database tools API] 790
- CONTINUE_AFTER_RAISERROR option
 - Open Client 6
- continue_download a_bit_fielda_sync_db structure [database tools API] 790
- conversion
 - data types 434
- convert_value method
 - a_v4_extfn_proc_context 107
- cookies
 - creating 647
 - session management 648
- Count propertySAErrorCollection class [SQL Anywhere .NET API] 276
- Count propertySAParameterCollection class [SQL Anywhere .NET API] 299
- CREATE EXISTING TABLE statement 944
 - proxy tables 964
- CREATE PROCEDURE statement
 - embedded SQL 463
- CREATE SERVER statement
 - syntax 966
- CREATE TABLE statement
 - syntax 968
- create_dnl_d_file const char *a_sync_db structure [database tools API] 790
- created_version chara_db_version_info structure [database tools API] 770
- CreateParameter method
 - using 133
- creating
 - proxy tables 964
 - web services 631
- cross site scripting
 - web services 653
- CS-Library
 - about 617
- ct_command function
 - describing results in Open Client 622
 - executing statements in Open Client 620
- ct_cursor function
 - Open Client 621
- ct_dynamic function
 - Open Client 621
- ct_results function
 - Open Client 622
- ct_send function
 - Open Client 622
- CUBE operation 845, 846, 855
 - example 857
 - NULL 848
 - SELECT statement 855
- Current propertyDREnumerator class [SQL Anywhere .NET API] 271
- current row 866
- CURRENT ROW 863, 864
- cursor positioning
 - troubleshooting 138
- cursor sensitivity and performance
 - about 153
- cursors
 - about 135
 - ADO 158
 - ADO.NET 158
 - asensitive 150
 - availability 142
 - benefits 136
 - block cursors 143
 - canceling 142
 - db_cancel_request function 473
 - delete 622

Index

- describing result sets 160
- determining what cursors exist for a connection
 - 135
- dynamic 149
- DYNAMIC SCROLL and asensitive cursors
 - 150
- DYNAMIC SCROLL and cursor positioning
 - 138
- embedded SQL supported types 159
- embedded SQL usage 453
- example C code 419
- fat 139
- fetching multiple rows 139
- fetching rows 138
- insensitive 148
- inserting multiple rows 140
- inserting rows 140
- internals 143
- isolation level 138
- keyset-driven 151
- limitations 138
- membership 144
- ODBC and SAP Sybase IQ types 158
- OLE DB 158
- Open Client 621
- order 144
- platforms 142
- positioning 138
- prefetch performance 154
- prepared statements 137
- properties 142
- Python 548
- read-only 148
- requesting 158
- result sets 135
- savepoints 164
- SCROLL 151
- scrollability 142
- scrollable 139
- sensitive 149
- sensitivity 142
- sensitivity and deletion example 145
- sensitivity and isolation levels 157
- sensitivity and performance 153
- sensitivity and update example 146
- sensitivity in SAP Sybase IQ 143
- sensitivity overview 144
- static 148
- stored procedures 464

- transactions 163
- uniqueness 142
- unspecified sensitivity 150
- updatability 142
- update 622
- updating 325
- updating and deleting rows 140
- uses 135
- using 137
- value-sensitive 151
- values 144
- viewing contents of cursors for a connection
 - 135
- visible changes 144
- work tables 153

cursors and bookmarks

- about 143

D

- data
 - accessing with the .NET Data Provider 170
 - manipulating with the .NET Data Provider
 - 170
- data access
 - OLE DB 320
- data connection
 - Visual Studio 203
- data type conversions
 - indicator variables 434
- data types
 - C data types 427
 - dynamic SQL 443
 - embedded SQL 423
 - host variables 427
 - in web services handlers 680
 - Open Client mapping 619
 - Open Client ranges 619
 - SQLDA 445
- data_size size_ta_sqlany_data_info structure [SQL Anywhere C API] 531
- data_store_type const char *a_create_db structure [database tools API] 761
- DataAdapter
 - about 170
 - deleting data 175
 - inserting data 175
 - obtaining primary key values 182
 - retrieving data 177, 178
 - updating data 175

- DataAdapter propertySACommandBuilder class [SQL Anywhere .NET API] 243
- database management
 - dbtools 731
- database options
 - Open Client 6
 - set for jConnect 384
- database properties
 - db_get_property function 476
- database servers
 - functions 486
- database tools API a_backup_db structure 751
- database tools API a_change_log structure 756
- database tools API a_create_db structure 759
- database tools API a_db_info structure 765
- database tools API a_db_version_info structure 770
- database tools API a_dblic_info structure 771
- database tools API a_dbtools_info structure 773
- database tools API a_log_file_info structure 773
- database tools API a_name structure 775
- database tools API a_remote_sql structure 775
- database tools API a_sync_db structure 787
- database tools API a_syncpub structure 808
- database tools API a_sysinfo structure 809
- database tools API a_table_info structure 810
- database tools API a_translate_log structure 812
- database tools API a_truncate_log structure 821
- database tools API a_validate_db structure 822
- database tools API an_erase_db structure 824
- database tools API an_unload_db structure 826
- database tools API an_upgrade_db structure 839
- Database Tools C API 739
- database tools interface
 - about 731
- database tools library
 - about 731
- DatabaseFile propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 250
- DatabaseKey propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 250
- DatabaseName
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 250
- databases
 - installing jConnect metadata support 382
 - multiple on server 9
 - proxy 5
 - storing Java classes 369
- URL 383
- DatabaseSwitches
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 250
- datagrid control
 - Visual Studio 207
- DataSet
 - SAP Sybase IQ .NET Data Provider 175
- DataSourceInformation
 - fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 285
- DataSourceName
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 250
- DataTypes fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 285
- DATETIME data type
 - embedded SQL 427
 - Open client conversion 619
- DB_ACTIVE_CONNECTION
 - db_find_engine function 475
- db_backup function
 - about 468
 - dbbackup utility 467
- DB_BACKUP_CLOSE_FILE parameter
 - about 468
- DB_BACKUP_END parameter
 - about 468
- DB_BACKUP_INFO parameter
 - about 468
- DB_BACKUP_INFO_CHKPT_LOG parameter
 - about 468
- DB_BACKUP_INFO_PAGES_IN_BLOCK parameter
 - about 468
- DB_BACKUP_OPEN_FILE parameter
 - about 468
- DB_BACKUP_PARALLEL_READ parameter
 - about 468
- DB_BACKUP_PARALLEL_START parameter
 - about 468
- DB_BACKUP_READ_PAGE parameter
 - about 468
- DB_BACKUP_READ_RENAME_LOG parameter
 - about 468
- DB_BACKUP_START parameter
 - about 468
- DB_CALLBACK_CONN_DROPPED 481

Index

- DB_CALLBACK_CONN_DROPPED callback parameter 481
- DB_CALLBACK_DEBUG_MESSAGE 481
- DB_CALLBACK_DEBUG_MESSAGE callback parameter 481
- DB_CALLBACK_FINISH 481
- DB_CALLBACK_FINISH callback parameter 481
- DB_CALLBACK_MESSAGE 482
- DB_CALLBACK_MESSAGE callback parameter 482
- DB_CALLBACK_START 481
- DB_CALLBACK_START callback parameter 481
- DB_CALLBACK_VALIDATE_FILE_TRANSFERENCE 482
- DB_CALLBACK_VALIDATE_FILE_TRANSFERENCE callback parameter 482
- DB_CALLBACK_WAIT 481
- DB_CALLBACK_WAIT callback parameter 481
- DB_CAN_MULTI_CONNECT
 - db_find_engine function 475
- DB_CAN_MULTI_DB_NAME
 - db_find_engine function 475
- db_cancel_request function
 - about 473
 - request management 466
- db_change_char_charset function
 - about 473
- db_change_nchar_charset function
 - about 474
- DB_CLIENT
 - db_find_engine function 475
- DB_CONNECTION_DIRTY
 - db_find_engine function 475
- DB_DATABASE_SPECIFIED
 - db_find_engine function 475
- DB_ENGINE
 - db_find_engine function 475
- db_find_engine function
 - about 475
- db_fini function
 - about 475
- db_fini_dll
 - calling 418
- db_get_property function
 - about 476
- db_init function
 - about 477
- db_init_dll
 - calling 418
- db_is_working function
 - about 477
 - request management 466
- db_locate_servers function
 - about 478
- db_locate_servers_ex function
 - about 479
- DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
 - about 479
- DB_LOOKUP_FLAG_DATABASES
 - about 479
- DB_LOOKUP_FLAG_NUMERIC
 - about 479
- DB_NO_DATABASES
 - db_find_engine function 475
- DB_PROP_CLIENT_CHARSET
 - usage 476
- DB_PROP_DBLIB_VERSION
 - usage 476
- DB_PROP_SERVER_ADDRESS
 - usage 476
- db_register_a_callback function
 - about 480
 - request management 466
- db_size unsigned int a_create_db structure [database tools API] 761
- db_size_unit int a_create_db structure [database tools API] 761
- db_start_database function
 - about 483
- db_start_engine function
 - about 484
- db_stop_database function
 - about 485
- db_stop_engine function
 - about 485
- db_string_connect function
 - about 486
- db_string_disconnect function
 - about 487
- db_string_ping_server function
 - about 487
- db_time_change function
 - about 488
- DB-Library
 - about 617

- dba_pwd char *a_create_db structure [database tools API] 761
- dba_uid char *a_create_db structure [database tools API] 761
- dbbufsize unsigned shorta_db_info structure [database tools API] 767
- DbConnection propertySACommand class [SQL Anywhere .NET API] 236
- DbConnection propertySATransaction class [SQL Anywhere .NET API] 316
- DBD::SQLAnywhere
 - about 535
 - connecting to a database 539
 - executing SQL statements 540
 - handling multiple result sets 541
 - inserting rows 542
 - writing Perl scripts 538
- dbdata.dll
 - SAP Sybase IQ .NET Data Provider 195
- DBLIB
 - dynamic loading 418
 - interface library 409
- dbname const char *a_change_log structure [database tools API] 756
- dbname const char *a_create_db structure [database tools API] 761
- dbname const char *a_log_file_info structure [database tools API] 774
- dbname const char *an_erase_db structure [database tools API] 824
- dbnamebuffer char *a_db_info structure [database tools API] 767
- dbodbc16.dll
 - linking 340
- DbParameterCollection propertySACommand class [SQL Anywhere .NET API] 236
- DbProviderFactory
 - registering 195
- dbtool16.dll
 - about 731
- DBTools interface
 - about 731
 - calling DBTools functions 733
 - example program 736
 - finalizing 732
 - finishing 732
 - initializing 732
 - introduction 731
 - return codes 738
 - starting 732
 - using 731
- DbTransaction propertySACommand class [SQL Anywhere .NET API] 237
- DbType propertySAParameter class [SQL Anywhere .NET API] 290
- dbupgrad utility
 - installing jConnect metadata support 382
- debug a_bit_fielda_remote_sql structure [database tools API] 777
- debug a_bit_fielda_sync_db structure [database tools API] 790
- debug a_bit_fielda_translate_log structure [database tools API] 813
- debug a_bit_fieldan_unload_db structure [database tools API] 827
- debug_dump_char a_bit_fielda_sync_db structure [database tools API] 790
- debug_dump_char a_bit_fielda_translate_log structure [database tools API] 813
- debug_dump_hex a_bit_fielda_sync_db structure [database tools API] 790
- debug_dump_hex a_bit_fielda_translate_log structure [database tools API] 813
- debug_dump_size a_sql_uint32a_remote_sql structure [database tools API] 778
- debug_dump_size a_sql_uint32a_sync_db structure [database tools API] 791
- debug_dump_size a_sql_uint32a_translate_log structure [database tools API] 814
- debug_page_offsets a_bit_fielda_remote_sql structure [database tools API] 778
- debug_page_offsets a_bit_fielda_sync_db structure [database tools API] 791
- debug_page_offsets a_bit_fielda_translate_log structure [database tools API] 814
- debug_sql_remote a_bit_fielda_translate_log structure [database tools API] 814
- DECIMAL data type
 - embedded SQL 427
- DECL_BIGINT macro
 - about 427
- DECL_BINARY macro
 - about 427
- DECL_BIT macro
 - about 427
- DECL_DATETIME macro
 - about 427

Index

- DECL_DECIMAL macro
 - about 427
- DECL_FIXCHAR macro
 - about 427
- DECL_LONGBINARy macro
 - about 427
- DECL_LONGNVARCHAR macro
 - about 427
- DECL_LONGVARCHAR macro
 - about 427
- DECL_NCHAR macro
 - about 427
- DECL_NFIXCHAR macro
 - about 427
- DECL_NVARCHAR macro
 - about 427
- DECL_UNSIGNED_BIGINT macro
 - about 427
- DECL_VARCHAR macro
 - about 427
- declaration section
 - about 427
- DECLARE section
 - about 427
- DECLARE statement
 - using cursors in embedded SQL 453
- declaring
 - embedded SQL data types 423
 - host variables 427
- default_collation chara_sysinfo structure [database tools API] 810
- default_collation const char *a_create_db structure [database tools API] 762
- default_window_title char *a_remote_sql structure [database tools API] 778
- default_window_title char *a_sync_db structure [database tools API] 791
- DELETE statement
 - JDBC 393
 - positioned 140
- DeleteCommand propertySADDataAdapter class [SQL Anywhere .NET API] 266
- deleted_a_bit_fielda_remote_sql structure [database tools API] 778
- DeleteDynamic method
 - JDBCExample 397
- DeleteStatic method
 - JDBCExample 394
- deltas between adjacent rows, computing 869
- DENSE_RANK function 873
- deploying
 - SAP Sybase IQ .NET Data Provider
 - applications 194
 - deploying the SAP Sybase IQ .NET Data Provider
 - about 194
- descending order 868
- describe
 - return value 93
- DESCRIBE SELECT LIST statement
 - dynamic SELECT statement 442
- DESCRIBE statement 445
 - multiple result sets 466
 - SQLDA fields 445
 - sqllen field 446
 - sqltype field 446
 - used in dynamic SELECT statements 442
- describe_column_get 26
 - attributes 26
- describe_column_set 40
 - attributes 41
- describe_parameter_get 55
- describe_parameter_set 72
- describe_udf_get 86
 - attributes 87
- describe_udf_set 88
- describing
 - NCHAR columns in Embedded SQL 446
 - result sets 160
- descriptors
 - describing result sets 160
- DesignTimeVisible propertySACCommand class [SQL Anywhere .NET API] 237
- DestinationColumn
 - propertySABulkCopyColumnMapping class [SQL Anywhere .NET API] 217
- DestinationOrdinal
 - propertySABulkCopyColumnMapping class [SQL Anywhere .NET API] 217
- DestinationOrdinalComparer class [SQL Anywhere .NET API] description 219, 222
- DestinationTableName propertySABulkCopy class [SQL Anywhere .NET API] 215
- developing applications with the .NET Data Provider
 - about 165
- DirectConnect 943, 945
- DirectConnect for Oracle 944

- direction
 - a_sqlany_data_directional_sqlany_bind_param structure [SQL Anywhere C API] 528
- direction
 - a_sqlany_data_directional_sqlany_bind_param_info structure [SQL Anywhere C API] 529
- Direction propertySAPParameter class [SQL Anywhere .NET API] 291
- DisableMultiRowFetch
 - propertySAPConnectionStringBuilder class [SQL Anywhere .NET API] 251
- DISH services
 - .NET tutorial 709
 - about 629
 - commenting 634
 - creating 633
 - dropping 634
 - homogeneous 634
 - JAX-WS tutorial 715
 - SAP Sybase IQ web client tutorial 701
- display_create_a_bit_fieldan_unload_db structure [database tools API] 827
- display_create_dbinit_a_bit_fieldan_unload_db structure [database tools API] 827
- Distributed Transaction Coordinator
 - three-tier computing 727
- distributed transaction processing
 - using the SAP Sybase IQ .NET Data Provider 186
- distributed transactions
 - about 725
 - architecture 728
 - enlistment 727
 - recovery 729
 - restrictions 728
 - three-tier computing 726
- distribution functions 843, 860, 886
- dl_insert_width_a_sql_uint32a_sync_db structure [database tools API] 791
- dl_use_put_a_bit_fielda_sync_db structure [database tools API] 791
- dlg_info_msg_a_sql_uint32a_sync_db structure [database tools API] 791
- DLL entry points
 - about 467
- dll_handle void *SQLAnywhereInterface structure [SQL Anywhere C API] 521
- DllMain
 - calling db_fini 475
- DLLs
 - multiple SQLCAs 439
- dnld_fail_len_a_sql_uint32a_sync_db structure [database tools API] 791
- dnld_file_extra const char *a_sync_db structure [database tools API] 792
- dnld_gen_num_a_bit_fielda_sync_db structure [database tools API] 792
- dnld_read_size_a_sql_uint32a_sync_db structure [database tools API] 792
- DoBroadcast propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 312
- download_only_a_bit_fielda_sync_db structure [database tools API] 792
- DREnumerator class [SQL Anywhere .NET API]
 - Current property 271
- DREnumerator class [SQL Anywhere .NET API]
 - description 270
- driver load error 943
- drivers
 - jConnect JDBC driver 378
 - linking the SAP Sybase IQ ODBC driver on Windows 340
 - SQL Anywhere JDBC driver 378
- DROP SERVER statement
 - syntax 985
- dropping
 - web services 634
- DT_BIGINT embedded SQL data type 423
- DT_BINARY embedded SQL data type 425
- DT_BIT embedded SQL data type 423
- DT_DATE embedded SQL data type 424
- DT_DECIMAL embedded SQL data type 424
- DT_DOUBLE embedded SQL data type 423
- DT_FIXCHAR embedded SQL data type 424
- DT_FLOAT embedded SQL data type 423
- DT_HAS_USERTYPE_INFO 445
- DT_INT embedded SQL data type 423
- DT_LONGBINARY embedded SQL data type 425
- DT_LONGNVARCHAR embedded SQL data type 425
- DT_LONGVARCHAR embedded SQL data type 424
- DT_NFIXCHAR embedded SQL data type 424
- DT_NSTRING embedded SQL data type 424
- DT_NVARCHAR embedded SQL data type 424

Index

- DT_PROCEDURE_IN
 - using 466
- DT_PROCEDURE_OUT
 - using 466
- DT_SMALLINT embedded SQL data type 423
- DT_STRING embedded SQL data type 424
- DT_TIME embedded SQL data type 424
- DT_TIMESTAMP embedded SQL data type 424
- DT_TIMESTAMP_STRUCT embedded SQL data type 425
- DT_TINYINT embedded SQL data type 423
- DT_UNSBIGINT embedded SQL data type 423
- DT_UNSENT embedded SQL data type 423
- DT_UNSMALLINT embedded SQL data type 423
- DT_VARCHAR embedded SQL data type 424
- DT_VARIABLE embedded SQL data type 426
- DTC
 - isolation levels 729
 - three-tier computing 727
- DTC isolation levels
 - about 729
- dynamic cursors
 - about 149
 - ODBC 158
 - sample 421
- DYNAMIC SCROLL cursors
 - asensitive cursors 150
 - embedded SQL 159
 - troubleshooting 138
- dynamic SELECT statement
 - DESCRIBE SELECT LIST statement 442
- dynamic SQL
 - about 440
 - SQLDA 443
- E**
- EAServer
 - three-tier computing 727
- Elevate propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 251
- embedded SQL
 - about 409
 - authorization 411
 - autocommit mode 161
 - calling db_fini from DIIMain 475
 - character strings 411
 - compile and link process 410
 - controlling autocommit behavior 161
 - cursor examples 419
 - cursor types 142
 - cursors 159
 - development 409
 - dynamic cursors 421
 - dynamic statements 440
 - example program 417
 - FETCH FOR UPDATE 155
 - fetching data 452
 - functions 467
 - header files 416
 - host variables 426
 - import libraries 416
 - line numbers 411
 - SQL statements 131
 - statement summary 493
 - static statements 440
 - using cursors 453
- embedded SQL data type 424
- emergency server shutdown 959
- encoding const char *a_create_db structure [database tools API] 762
- encrypt a_bit_fielda_create_db structure [database tools API] 762
- encrypted_stream_opts const char *a_sync_db structure [database tools API] 792
- encrypted_tables a_bit_fielda_create_db structure [database tools API] 762
- encrypted_tables a_bit_fielda_db_info structure [database tools API] 767
- encrypted_tables a_bit_fieldan_unload_db structure [database tools API] 827
- EncryptedPassword
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 251
- encryption a_bit_fielda_sysinfo structure [database tools API] 810
- Encryption propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 251
- encryption_algorithm const char *a_create_db structure [database tools API] 762
- encryption_algorithm const char *an_unload_db structure [database tools API] 827
- encryption_key char *a_change_log structure [database tools API] 757
- encryption_key char *a_remote_sql structure [database tools API] 778
- encryption_key char *a_sync_db structure [database tools API] 793

- encryption_key const char *a_create_db structure [database tools API] 763
- encryption_key const char *a_log_file_info structure [database tools API] 774
- encryption_key const char *a_translate_log structure [database tools API] 814
- encryption_key const char *an_erase_db structure [database tools API] 824
- encryption_key const char *an_unload_db structure [database tools API] 828
- Enlist propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 252
- enlistment
 - distributed transactions 727
- entered_dialog a_bit_fielda_sync_db structure [database tools API] 793
- Enterprise Connect Data Access 943, 944
- Entity Framework
 - using 188
- Entity Framework support
 - iAnywhere.Data.SQLAnywhere provider 166
- entry points
 - calling DBTools functions 733
- enumerated type
 - a_v4_extfn_describe_col_type 90
 - a_v4_extfn_describe_parm_type 91
 - a_v4_extfn_describe_return 93
 - a_v4_extfn_describe_udf_type 95
 - a_v4_extfn_partitionby_col_num 114
 - a_v4_extfn_state 95
- erase a_bit_fieldan_erase_db structure [database tools API] 825
- error codes
 - SAP Sybase IQ exit codes 738
- error handling
 - Java 370
 - SAP Sybase IQ .NET Data Provider 187
- error messages
 - embedded SQL function 493
- errortn MSG_CALLBACKa_backup_db structure [database tools API] 753
- errortn MSG_CALLBACKa_change_log structure [database tools API] 757
- errortn MSG_CALLBACKa_create_db structure [database tools API] 763
- errortn MSG_CALLBACKa_db_info structure [database tools API] 767
- errortn MSG_CALLBACKa_db_version_info structure [database tools API] 770
- errortn MSG_CALLBACKa_dblic_info structure [database tools API] 771
- errortn MSG_CALLBACKa_dbtools_info structure [database tools API] 773
- errortn MSG_CALLBACKa_log_file_info structure [database tools API] 774
- errortn MSG_CALLBACKa_remote_sql structure [database tools API] 778
- errortn MSG_CALLBACKa_sync_db structure [database tools API] 793
- errortn MSG_CALLBACKa_translate_log structure [database tools API] 814
- errortn MSG_CALLBACKa_truncate_log structure [database tools API] 821
- errortn MSG_CALLBACKa_validate_db structure [database tools API] 823
- errortn MSG_CALLBACKan_erase_db structure [database tools API] 825
- errortn MSG_CALLBACKan_unload_db structure [database tools API] 828
- errortn MSG_CALLBACKan_upgrade_db structure [database tools API] 839
- errors 435
 - HTTP codes 695
 - SOAP faults 695
- Errors propertySAException class [SQL Anywhere .NET API] 277
- Errors propertySAInfoMessageEventArgs class [SQL Anywhere .NET API] 283
- escape syntax
 - Interactive SQL 405
- escape_char charan_unload_db structure [database tools API] 828
- escape_char_present a_bit_fieldan_unload_db structure [database tools API] 828
- esql.dll.c
 - about 418
- est_upld_row_cnt a_sql_uint32a_sync_db structure [database tools API] 793
- evaluate_extfn 98
- examples
 - OLAP 902
- exceptions
 - Java 370
 - SAP Sybase IQ .NET Data Provider 187
- exclude_foreign_keys a_bit_fieldan_unload_db structure [database tools API] 828
- exclude_hooks a_bit_fieldan_unload_db structure [database tools API] 829

Index

- exclude_procedures a_bit_fieldan_unload_db
 - structure [database tools API] 829
- exclude_tables a_bit_fieldan_unload_db structure
 - [database tools API] 829
- exclude_triggers a_bit_fieldan_unload_db structure
 - [database tools API] 829
- exclude_views a_bit_fieldan_unload_db structure
 - [database tools API] 829
- EXEC SQL
 - embedded SQL development 418
- execute method
 - Python 548
- EXECUTE statement
 - stored procedures in embedded SQL 463
 - using 440
- executeBatch
 - PreparedStatement class 398
 - Statement class 393
- executemany method
 - Python 549
- ExecuteNonQuery method
 - using SACommand 172
- ExecuteReader method
 - ADO.NET prepared statements 133
 - using SACommand 171
- ExecuteScalar method
 - using SACommand 171
- executeUpdate
 - Statement class 393
- executeUpdate JDBC method
 - using 134
- executing SQL statements
 - in applications 131
- execution phase
 - a_v4_extfn_state enumerator 95
- exename char *a_dblic_info structure [database tools API] 772
- exit codes
 - about 738
- EXP function 896
- exponential function 896
- exports file
 - dblib.def 416
- ext_opt char *a_syncpub structure [database tools API] 809
- extended_options char *a_sync_db structure
 - [database tools API] 793
- extensions to GROUP BY clause 843, 845
- external function
 - a_v4_extfn_proc 97
- external logins
 - about 948
- external procedure context
 - a_v4_extfn_proc_context 100
 - alloc method 109
 - close_result_set method 110
 - get_option method 108
 - open_result_set method 110
 - set_cannot_be_distributed 112
- EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL
 - get 30
 - set 45
- EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE
 - get 34
 - set 48
- EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES
 - set 30, 46
- EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT
 - get 33
 - set 48
- EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE
 - get 32
 - set 47
- EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER
 - get 35
 - set 49
- EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE
 - get 38
 - set 52
- EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE
 - get 36
 - set 50
- EXTFNAPIV4_DESCRIBE_COL_NAME
 - set 27, 41
- EXTFNAPIV4_DESCRIBE_COL_SCALE
 - get 29
 - set 44
- EXTFNAPIV4_DESCRIBE_COL_TYPE
 - get 27
 - set 42

- EXTFNAPIV4_DESCRIBE_COL_VALUES_SU
BSET_OF_INPUT
 - get 40
 - set 54
 - EXTFNAPIV4_DESCRIBE_COL_WIDTH
 - set 28, 43
 - EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_N
ULL
 - get 60, 61
 - set 77
 - EXTFNAPIV4_DESCRIBE_PARM_CONSTANT
_VALUE
 - get 65
 - set 78
 - EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_
VALUES
 - get 62
 - set 77
 - EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTA
NT
 - get 64
 - set 78
 - EXTFNAPIV4_DESCRIBE_PARM_NAME
 - get 56
 - set 74
 - EXTFNAPIV4_DESCRIBE_PARM_SCALE
 - get 59
 - set 76
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_HA
S_REWIND
 - get 70
 - set 84
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_NU
M_COLUMNS
 - get 66
 - set 79
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_NU
M_ROWS
 - get 66
 - set 80
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_OR
DERBY
 - get 67
 - set 81
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_PA
RTITIONBY
 - get 68
 - set 82
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_RE
QUEST_REWIND
 - get 70
 - set 83
 - EXTFNAPIV4_DESCRIBE_PARM_TABLE_UN
USED_COLUMNS
 - get 71
 - set 85
 - EXTFNAPIV4_DESCRIBE_PARM_TYPE
 - get 57
 - set 74
 - EXTFNAPIV4_DESCRIBE_PARM_WIDTH
 - get 57
 - set 75
 - EXTFNAPIV4_DESCRIBE_UDF_NUM_PARM
S
 - get 87
 - set 89
 - extra_audit a_bit_fielda_translate_log structure
[database tools API] 814
 - extract a_bit_fieldan_unload_db structure [database
tools API] 830
- ## F
- fat cursors
 - about 139
 - FETCH FOR UPDATE
 - embedded SQL 155
 - ODBC 155
 - fetch operation
 - cursors 138
 - multiple rows 139
 - scrollable cursors 139
 - FETCH statement
 - dynamic queries 442
 - multi-row 456
 - using 452
 - using cursors in embedded SQL 453
 - wide 456
 - fetch_block
 - v4 API method 121
 - fetch_into
 - v4 API method 119
 - fetchall method
 - Python 548
 - fetches
 - array fetches 456
 - wide fetches 456

Index

- fetching
 - embedded SQL 452
 - limits 138
 - file transfer 482
 - file_size a_sql_uint32a_db_info structure [database tools API] 767
 - FileDataSourceName
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 252
 - filename const char *a_db_version_info structure [database tools API] 770
 - fill_s_sqlda function
 - about 488
 - fill_sqlda function
 - about 489
 - fill_sqlda_ex function
 - about 489
 - FLOOR function 897
 - force_chaining a_bit_fielda_translate_log structure [database tools API] 814
 - force_recovery a_bit_fielda_translate_log structure [database tools API] 815
 - ForceStart connection parameter
 - db_start_engine 484
 - ForceStart propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 252
 - foreign keys
 - integrity constraints 968
 - unnamed 968
 - ForeignKeys fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 285
 - free_filled_sqlda function
 - about 490
 - free_pages a_sql_uint32a_db_info structure [database tools API] 767
 - free_sqlda function
 - about 491
 - free_sqlda_noinf function
 - about 491
 - frequency a_sql_uint32a_remote_sql structure [database tools API] 779
 - full_q_scan a_bit_fielda_remote_sql structure [database tools API] 779
 - functions
 - aggregate 859
 - analytical 843, 858
 - BIT_LENGTH function 894
 - calling DBTools functions 733
 - CEIL function 895
 - CEILING function 895
 - correlation 882
 - covariance 882, 883
 - DENSE_RANK function 873
 - distribution 843, 886
 - embedded SQL 467
 - EXP function 896
 - FLOOR function 897
 - inverse distribution 886
 - LENGTH function 898
 - numeric 843
 - numerical 892
 - ordered sets 886
 - PERCENT_RANK function 875
 - PERCENTILE_CONT function 886, 888
 - PERCENTILE_DISC function 886, 890
 - POWER function 898
 - RANK function 871
 - ranking 843, 870
 - reporting 880
 - requirements for web clients 662
 - SAP Sybase IQ PHP module 566
 - simple aggregate 859
 - SQRT function 899
 - standard deviation 881
 - statistical 843
 - statistical aggregate 881
 - STDDEV_POP function 882
 - STDDEV_SAMP function 882
 - VAR_POP function 882
 - VAR_SAMP function 882
 - variance 881
 - web clients 662
 - WIDTH_BUCKET function 899
 - window 844, 880
 - windowing 859
 - windowing aggregate 843, 880
- ## G
- generate_reciprocals a_bit_fielda_translate_log structure [database tools API] 815
 - generation_number unsigned shorta_change_log structure [database tools API] 757
 - genscript a_bit_fieldan_unload_db structure [database tools API] 830
 - get_blob method
 - a_v4_extfn_proc_context 111
 - a_v4_extfn_table_context 124

- get_is_cancelled method
 - a_v4_extfn_proc_context 105
 - get_option
 - v4 API method 108
 - get_value method
 - a_v4_extfn_proc_context 102
 - get_value_is_constant method
 - a_v4_extfn_proc_context 104
 - getAutoCommit method 390
 - GetBytes method
 - using 184
 - GetChars method
 - using 184
 - getConnection method 390
 - GetSchemaTable method
 - using SADataReader 172
 - GetTimeSpan method
 - using 184
 - getUpdateCounts
 - BatchUpdateException 393
 - GNU compiler
 - embedded SQL support 415
 - GRANT statement
 - JDBC 400
 - GROUP BY
 - clause extensions 845
 - CUBE 846
 - ROLLUP 846
 - GROUP BY clause extensions 845
 - GROUPING function
 - NULL 848
 - ROLLUP operation 848
- H**
- Hadoop 13
 - HEADER clause
 - managing 666
 - header files
 - embedded SQL 416
 - headers
 - accessing in HTTP web services 639
 - in SOAP web services 643
 - hide_conn_str a_bit_fielda_sync_db structure
 - [database tools API] 794
 - hide_ml_pwd a_bit_fielda_sync_db structure
 - [database tools API] 794
 - History() enumeration [database tools API] 748
 - Host
 - accessing HTTP headers 641
 - Host propertySAConnectionStringBuilder class
 - [SQL Anywhere .NET API] 252
 - Host propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 312
 - host variables
 - about 426
 - data types 427
 - declaring 427
 - example 640
 - not supported in batches 426
 - SQLDA 445
 - uses 431
 - hotlog_filename const char *a_backup_db structure
 - [database tools API] 753
 - hovering_frequency a_sql_uint32a_sync_db structure [database tools API] 794
 - HTML services
 - about 629
 - commenting 634
 - creating 631
 - dropping 634
 - quick start 660
 - quick start for web clients 658
 - quick start for web servers 625
 - HTTP headers
 - accessing 641
 - HTTP protocol
 - configuring 627
 - enabling 626
 - HTTP request headers
 - accessing 641
 - management 666
 - HTTP requests
 - structures 693
 - HTTP system procedures
 - alphabetical list 653
 - HTTP_HEADER function
 - example 640
 - HTTP_VARIABLE function
 - example 640
 - HttpMethod
 - accessing HTTP headers 641
 - HttpQueryString
 - accessing HTTP headers 641
 - HTTPS protocol
 - configuring 627
 - enabling 626
 - HttpStatus
 - accessing HTTP headers 641

Index

HttpURI

accessing HTTP headers 641

HttpVersion

accessing HTTP headers 641

I

iAnywhere.Data.SQLAnywhere

Entity Framework support 166

identifiers

needing quotes 491

IdleTimeout propertySACConnectionStringBuilder

class [SQL Anywhere .NET API] 252

ignore_dbsync_trunc a_bit_fielda_change_log

structure [database tools API] 757

ignore_debug_interrupt a_bit_fielda_sync_db

structure [database tools API] 794

ignore_hook_errors a_bit_fielda_sync_db structure

[database tools API] 794

ignore_hovering a_bit_fielda_sync_db structure

[database tools API] 795

ignore_ltm_trunc a_bit_fielda_change_log

structure [database tools API] 757

ignore_remote_trunc a_bit_fielda_change_log

structure [database tools API] 757

ignore_scheduling a_bit_fielda_sync_db structure

[database tools API] 795

import libraries

alternatives 418

DBTools 732

embedded SQL 416

introduction 410

ODBC 340

import statement

jConnect 382

INCLUDE statement

SQLCA 435

include_audit a_bit_fielda_translate_log structure

[database tools API] 815

include_destination_sets const char

*a_translate_log structure [database tools API] 815

include_publications const char *a_translate_log

structure [database tools API] 815

include_scan_range char *a_remote_sql structure

[database tools API] 779

include_scan_range const char *a_sync_db

structure [database tools API] 795

include_scan_range const char *a_translate_log

structure [database tools API] 815

include_source_sets const char *a_translate_log

structure [database tools API] 815

include_subsets a_bit_fielda_translate_log

structure [database tools API] 816

include_tables const char *a_translate_log structure

[database tools API] 816

include_trigger_trans a_bit_fielda_translate_log

structure [database tools API] 816

include_where_subscribe a_bit_fieldan_unload_db

structure [database tools API] 830

index a_bit_fielda_validate_db structure [database

tools API] 823

index_pages a_sql_uint32a_table_info structure

[database tools API] 811

index_used a_sql_uint32a_table_info structure

[database tools API] 811

index_used_pct a_sql_uint32a_table_info structure

[database tools API] 811

IndexColumns fieldSAMetaDataCollectionNames

class [SQL Anywhere .NET API] 286

Indexes fieldSAMetaDataCollectionNames class

[SQL Anywhere .NET API] 286

indicator

wide fetch 456

indicator variables

about 432

data type conversion 434

NULL 433

SQLDA 445

summary of values 434

truncation 434

init_cache a_sql_uint32a_sync_db structure

[database tools API] 795

init_cache_suffix chara_sync_db structure

[database tools API] 795

initialized intSQLAnywhereInterface structure

[SQL Anywhere C API] 521

InitString propertySACConnectionStringBuilder

class [SQL Anywhere .NET API] 253

INOUT parameters

Java in the database 373

Inprocess option

Linked Server 330, 332

input_value

a_sqlany_data_valuea_sqlany_bind_para

m_info structure [SQL Anywhere C API]

529

insensitive cursors

about 148

- cursor properties 142
- delete example 145
- embedded SQL 159
- introduction 144
- update example 146
- INSERT statement
 - JDBC 393
 - performance 132
 - writing Python scripts 549
- InsertCommand propertySADDataAdapter class
 - [SQL Anywhere .NET API] 267
- InsertDynamic method
 - JDBCExample 395, 397
- inserting data
 - multi-row 456
 - wide inserts 456
- inserts
 - JDBC 398
- InsertStatic method
 - JDBCExample 393, 394
- installing
 - Java classes into a database 371
 - jConnect metadata support 382
- installkey char *a_dblic_info structure [database tools API] 772
- Instance fieldSAFactory class [SQL Anywhere .NET API] 282
- Instance propertySADDataSourceEnumerator class
 - [SQL Anywhere .NET API] 272
- instances 390
- Integrated propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 253
- Interactive SQL
 - JDBC escape syntax 405
- interface libraries
 - DBLIB 409
 - dynamic loading 418
- interfaces
 - SAP Sybase IQ embedded SQL 409
- interfaces file 945, 946
- Interop
 - web services 687
- inverse distribution functions 886
- IPV6 propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 312
- iq_params void *a_create_db structure [database tools API] 763
- iqsqlpp utility
 - about 411
 - preprocessor options 411
 - syntax 411
- is_null sacapi_bool *a_sqlany_data_value structure
 - [SQL Anywhere C API] 533
- is_null sacapi_bool *a_sqlany_data_info structure
 - [SQL Anywhere C API] 531
- IsFixedSize propertySAParameterCollection class
 - [SQL Anywhere .NET API] 299
- IsNull propertySAParameter class [SQL Anywhere .NET API] 291
- isolation level 390
 - readonly-statement-snapshot 164
 - snapshot 164
 - statement-snapshot 164
- isolation levels
 - ADO programming 326
 - applications 163
 - cursor sensitivity 157
 - cursors 138
 - DTC 729
 - lost updates 155
 - setting for the SATransaction object 186
- ISOLATION_LEVEL option
 - Open Client 6
- isolation_level unsigned short a_unload_db structure [database tools API] 830
- isolation_set a_bit_field a_unload_db structure [database tools API] 830
- IsolationLevel propertySATransaction class [SQL Anywhere .NET API] 317
- ISOLATIONLEVEL_BROWSE
 - about 729
- ISOLATIONLEVEL_CHAOS
 - about 729
- ISOLATIONLEVEL_CURSORSTABILITY
 - about 729
- ISOLATIONLEVEL_ISOLATED
 - about 729
- ISOLATIONLEVEL_READCOMMITTED
 - about 729
- ISOLATIONLEVEL_READUNCOMMITTED
 - about 729
- ISOLATIONLEVEL_REPEATABLE_READ
 - about 729
- ISOLATIONLEVEL_SERIALIZABLE
 - about 729
- ISOLATIONLEVEL_UNSPECIFIED
 - about 729

Index

- IsReadOnly property SAPParameterCollection class [SQL Anywhere .NET API] 300
- IsSynchronized property SAPParameterCollection class [SQL Anywhere .NET API] 300
- isysserver system table
 - remote servers for Component Integration Services 966
- J**
- JAR files
 - installing 371
- Java
 - in the database 369
 - JDBC 377
- Java class creation wizard
 - using 389
- Java in the database
 - about 369
 - error handling 370
 - installing classes 371
 - Java VM 370
 - key features 369
 - main method 371
 - NoSuchMethodException 372
 - returning result sets 372
 - security management 374
 - starting the VM 374
 - stopping the VM 374
 - storing classes 369
 - VM shutdown hooks 374
- Java stored procedures
 - about 372
 - example 372
- Java UDF 11, 12
- Java VM
 - shutdown hooks 374
 - starting 374
 - stopping 374
- JAX-WS
 - installing 719
 - tutorial 715
 - versions 719
- jconn4.jar
 - loading jConnect 383
 - loading jConnect JDBC driver 387
- jConnect
 - about 382
 - choosing a JDBC driver 378
 - CLASSPATH environment variable 382
 - connecting 383
 - database setup 382
 - download 382
 - external connections 384
 - installing metadata support 382
 - loading 383
 - packages 382
 - server-side connections 388
 - system objects 382
 - URL 383
 - versions supplied 382
- jconnect_a_bit_fielda_create_db structure [database tools API] 763
- jconnect_a_bit_fieldan_upgrade_db structure [database tools API] 839
- JDBC 390
 - about 377
 - applications overview 379
 - autocommit 390
 - autocommit mode 161
 - batched inserts 394
 - batched inserts example 398
 - client connections 384
 - client-side 380
 - connecting to a database 383
 - connection code 385
 - connections 380
 - controlling autocommit behavior 161
 - cursor types 142
 - data access 392
 - DELETE statement 393
 - escape syntax in Interactive SQL 405
 - example connection 384
 - example source code 378
 - INSERT statement 393
 - introduction to programming 377
 - jConnect 382
 - prepared statements 395
 - privileges 400
 - requirements 378
 - result sets 399
 - server-side 380
 - server-side connections 388
 - SQL Anywhere JDBC driver 380
 - SQL statements 131
 - UPDATE statement 393
 - ways to use 378
- JDBC callback
 - example 401

- JDBC defaults 390
 - JDBC drivers
 - choosing 378
 - compatibility 378
 - OSGi bundle 378
 - performance 378
 - JDBC escape syntax
 - using in Interactive SQL 405
 - JDBC transaction isolation level 390
 - JDBC-ODBC bridge
 - SQL Anywhere JDBC driver 378
 - JDBCExample class
 - about 392
 - JDBCExample.java
 - about 392
 - JSON services
 - about 629
 - commenting 634
 - creating 631
 - dropping 634
 - quick start 660
 - quick start for web clients 658
 - quick start for web servers 625
- K**
- Kerberos propertySACConnectionStringBuilder
 - class [SQL Anywhere .NET API] 253
 - Keys propertySACConnectionStringBuilderBase
 - class [SQL Anywhere .NET API] 253, 261, 312
 - keyset-driven cursors
 - ODBC 158
 - value-sensitive 151
 - kill_other_connections a_bit_fielda_sync_db
 - structure [database tools API] 796
- L**
- Language propertySACConnectionStringBuilder
 - class [SQL Anywhere .NET API] 254
 - last_upload_def a_syncpub *a_sync_db structure
 - [database tools API] 796
 - latest_backup a_bit_fielda_remote_sql structure
 - [database tools API] 779
 - LazyClose propertySACConnectionStringBuilder
 - class [SQL Anywhere .NET API] 254
 - LDAP propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 313
 - leave_output_on_error a_bit_fielda_translate_log
 - structure [database tools API] 816
 - LENGTH function 898
 - length of 435
 - length_size_t *a_sqlany_data_value structure [SQL Anywhere C API] 533
 - length SQLDA field
 - about 445
 - values 446
 - libdbtool16_r
 - about 731
 - libraries
 - dbtstm.lib 732
 - dbtool16.lib 732
 - embedded SQL 416
 - using the import libraries 732
 - library
 - dblib16.lib 416
 - dblibtm.lib 416
 - libdblib16_r.so 416
 - libdblib16.so 416
 - libdbtasks16_r.so 416
 - libdbtasks16.so 416
 - library functions
 - embedded SQL 467
 - licensing
 - web clients 646
 - line length
 - SQL preprocessor output 411
 - line numbers
 - SQL preprocessor utility (iqsqlpp) 411
 - link_debug a_bit_fielda_remote_sql structure
 - [database tools API] 779
 - Linked Servers
 - 4-part syntax 329
 - four-part syntax 329
 - Inprocess option 330, 332
 - OLE DB 329
 - openquery 329
 - RPC option 330, 332
 - RPC Out option 330, 332
 - security context 330, 332
 - LINQ support
 - .NET data provider features 166
 - LinqSample, .NET Data Provider sample project 167
 - LinqSample
 - .NET Data Provider sample project 167

Index

- lite_blob_handling a_bit_fielda_sync_db structure [database tools API] 796
- liveness 481
- LivenessTimeout
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 254
- locale char *a_remote_sql structure [database tools API] 779
- locale const char *an_unload_db structure [database tools API] 831
- LocalOnly propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 313
- log_file_name const char *a_remote_sql structure [database tools API] 780
- log_file_name const char *a_sync_db structure [database tools API] 796
- log_message method
 - a_v4_extfn_proc_context 107
- log_size a_sql_uint32a_remote_sql structure [database tools API] 780
- log_size a_sql_uint32a_sync_db structure [database tools API] 796
- logbufsize unsigned shorta_db_info structure [database tools API] 767
- LogFile propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 254
- logging
 - web services client information 694
- logical offset of a window frame 867
- logname char *a_log_file_info structure [database tools API] 774
- logname const char *a_change_log structure [database tools API] 758
- logname const char *a_create_db structure [database tools API] 763
- logname const char *a_translate_log structure [database tools API] 816
- logname_size size_ta_log_file_info structure [database tools API] 774
- lognamebuffer char *a_db_info structure [database tools API] 768
- logrtn MSG_CALLBACKa_remote_sql structure [database tools API] 780
- logrtn MSG_CALLBACKa_sync_db structure [database tools API] 797
- logrtn MSG_CALLBACKa_translate_log structure [database tools API] 816
- logs_dir const char *a_translate_log structure [database tools API] 817
- LONG BINARY data type
 - embedded SQL 460
 - retrieving in embedded SQL 461, 462
 - sending in embedded SQL 462, 463
- LONG NVARCHAR data type
 - embedded SQL 460
 - retrieving in embedded SQL 461, 462
 - sending in embedded SQL 462, 463
- LONG VARCHAR data type
 - embedded SQL 460
 - retrieving in embedded SQL 461, 462
 - sending in embedded SQL 462, 463
- LONGBINARY data type
 - embedded SQL 427
- LONGNVARCHAR data type
 - embedded SQL 427
- LONGVARCHAR data type
 - embedded SQL 427
- lost updates
 - about 155
- M**
- macros
 - _SQL_OS_WINDOWS 418
- main method
 - Java in the database 371
- make_auxiliary a_bit_fieldan_unload_db structure [database tools API] 831
- management tools
 - dbtools 731
- managing
 - transactions 955
- manual commit mode
 - controlling 161
 - implementation 162
 - transactions 161
- MapReduce 13
- match_mode a_bit_fielda_translate_log structure [database tools API] 817
- match_pos const char *a_translate_log structure [database tools API] 817
- max_cache a_sql_uint32a_sync_db structure [database tools API] 797
- max_cache_suffix chara_sync_db structure [database tools API] 797
- max_length a_sql_uint32a_remote_sql structure [database tools API] 780
- max_size size_ta_sqlany_column_info structure [SQL Anywhere C API] 530

- MaxPoolSize propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 254
- membership
 - result sets 144
- memory a_sql_uint32a_remote_sql structure [database tools API] 781
- Message propertySAError class [SQL Anywhere .NET API] 274
- Message propertySAException class [SQL Anywhere .NET API] 277
- Message propertySAInfoMessageEventArgs class [SQL Anywhere .NET API] 283
- messages 482
- MessageType propertySAInfoMessageEventArgs class [SQL Anywhere .NET API] 283
- metadata support
 - installing for jConnect 382
- MetaDataCollections
 - fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 286
- Microsoft SQL Server Management Studio
 - Linked Server 330, 332
- Microsoft Transaction Server
 - three-tier computing 727
- Microsoft Visual C++
 - embedded SQL support 415
- MIME
 - setting types 638
- MIME types
 - web services tutorial 697
- min_cache a_sql_uint32a_sync_db structure [database tools API] 797
- min_cache_suffix chara_sync_db structure [database tools API] 797
- MinPoolSize propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 255
- mirror_logs char *a_remote_sql structure [database tools API] 781
- mirrorbufsize unsigned shorta_db_info structure [database tools API] 768
- mirrorname char *a_log_file_info structure [database tools API] 774
- mirrorname const char *a_change_log structure [database tools API] 758
- mirrorname const char *a_create_db structure [database tools API] 763
- mirrorname_size size_ta_log_file_info structure [database tools API] 774
- mirrornamebuffer char *a_db_info structure [database tools API] 768
- mixed cursors
 - ODBC 158
- mlpassword char *a_sync_db structure [database tools API] 797
- MONEY data type
 - Open client conversion 619
- more a_bit_fielda_remote_sql structure [database tools API] 781
- MS SQL 943
- MS SQL Server 944, 945
- ms_filename const char *an_unload_db structure [database tools API] 831
- ms_reserve intan_unload_db structure [database tools API] 831
- ms_size intan_unload_db structure [database tools API] 831
- MSDASQL
 - OLE DB provider 319
- msgqueuern
 - MSG_QUEUE_CALLBACKa_remote_sql structure [database tools API] 781
- msgqueuern
 - MSG_QUEUE_CALLBACKa_sync_db structure [database tools API] 798
- msggrtn MSG_CALLBACKa_backup_db structure [database tools API] 753
- msggrtn MSG_CALLBACKa_change_log structure [database tools API] 758
- msggrtn MSG_CALLBACKa_create_db structure [database tools API] 764
- msggrtn MSG_CALLBACKa_db_info structure [database tools API] 768
- msggrtn MSG_CALLBACKa_db_version_info structure [database tools API] 771
- msggrtn MSG_CALLBACKa_dblic_info structure [database tools API] 772
- msggrtn MSG_CALLBACKa_remote_sql structure [database tools API] 781
- msggrtn MSG_CALLBACKa_sync_db structure [database tools API] 798
- msggrtn MSG_CALLBACKa_translate_log structure [database tools API] 817
- msggrtn MSG_CALLBACKa_truncate_log structure [database tools API] 821
- msggrtn MSG_CALLBACKa_validate_db structure [database tools API] 823

Index

- msgsrtn MSG_CALLBACKan_erase_db structure [database tools API] 825
- msgsrtn MSG_CALLBACKan_unload_db structure [database tools API] 831
- msgsrtn MSG_CALLBACKan_upgrade_db structure [database tools API] 840
- multi-row fetches
 - ESQL 456
- multi-row inserts
 - ESQL 456
- multi-row puts
 - ESQL 456
- multi-row queries
 - cursors 453
- multiple result sets
 - DESCRIBE statement 466
- multithreaded applications
 - embedded SQL 437
 - Java in the database 372
 - multiple SQLCAs in embedded SQL 439
- MyIP propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 313
- N**
- name char *a_sqlany_bind_param structure [SQL Anywhere C API] 528
- name char *a_sqlany_bind_param_info structure [SQL Anywhere C API] 529
- name char *a_sqlany_column_info structure [SQL Anywhere C API] 530
- name chara_name structure [database tools API] 775
- name SQLDA field
 - about 445
- NAMESPACE clause
 - managing 673
- namespaces
 - web services 687
- native_type
 - a_sqlany_native_typea_sqlany_column_info structure [SQL Anywhere C API] 530
- NativeError propertySAError class [SQL Anywhere .NET API] 274
- NativeError propertySAException class [SQL Anywhere .NET API] 278
- NativeError propertySAInfoMessageEventArgs class [SQL Anywhere .NET API] 284
- NCHAR data type
 - embedded SQL 427
- nchar_collation const char *a_create_db structure [database tools API] 764
- ncharcollationspecbuffer char *a_db_info structure [database tools API] 768
- ncharcollationspecbufsize unsigned shorta_db_info structure [database tools API] 768
- ncharencodingbuffer char *a_db_info structure [database tools API] 768
- ncharencodingbufsize unsigned shorta_db_info structure [database tools API] 769
- Network Service
 - OLE DB 329
- new_mlpassword char *a_sync_db structure [database tools API] 798
- NewPassword
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 255
- next struct a_name *a_name structure [database tools API] 775
- next struct a_syncpub *a_syncpub structure [database tools API] 809
- next struct a_table_info *a_table_info structure [database tools API] 811
- NEXT_CONNECTION function
 - example 651
- NEXT_HTTP_HEADER function
 - example 640
- NEXT_HTTP_VARIABLE function
 - example 640
- NO SCROLL cursors
 - about 148
 - embedded SQL 159
- no_confirm a_bit_fielda_backup_db structure [database tools API] 753
- no_confirm a_bit_fieldan_unload_db structure [database tools API] 832
- no_offline_logscan a_sql_uint32a_sync_db structure [database tools API] 798
- no_reload_status a_bit_fieldan_unload_db structure [database tools API] 832
- no_schema_cache a_bit_fielda_sync_db structure [database tools API] 799
- no_stream_compress a_bit_fielda_sync_db structure [database tools API] 799
- no_user_interaction a_bit_fielda_remote_sql structure [database tools API] 782
- nodecount a_sql_int32a_dblic_info structure [database tools API] 772

- NodeType propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 255
- notemp_size longan_unload_db structure [database tools API] 832
- NotifyAfter propertySABulkCopy class [SQL Anywhere .NET API] 216
- NuGet
 - Entity Framework 4 188
- NULL
 - CUBE operation 848
 - dynamic SQL 443
 - indicator variables 432
 - ROLLUP operation 848
- NULL values
 - example 848
- NULL values and subtotal rows 848
- null-terminated string 424
- nullable sacapi_boolany_sqlany_column_info structure [SQL Anywhere C API] 530
- numeric functions 843
 - CEIL 895
 - CEILING 895
 - EXP 896
 - FLOOR 897
 - POWER 898
 - SQRT 899
 - WIDTH_BUCKET 899
- NVARCHAR data type
 - embedded SQL 427
- O**
- obtaining time values
 - about 184
- ODBC
 - autocommit mode 161
 - controlling autocommit behavior 161
 - cursor types 142
 - cursors 158
 - driver name for SAP Sybase IQ 346
 - FETCH FOR UPDATE 155
 - import libraries 340
 - linking 340
 - sample program 343
 - SQL statements 131
- offline_dir const char *a_sync_db structure [database tools API] 799
- Offset propertySAPParameter class [SQL Anywhere .NET API] 291
- OLAP 11, 860
 - about 843
 - aggregate functions 859
 - analytical functions 843, 858
 - benefits 844
 - CUBE operation 855
 - current row 866
 - DENSE_RANK function 873
 - distribution functions 843, 860
 - extensions to GROUP BY clause 843
 - functionality 843
 - Grouping() 845
 - NULL values 848
 - numeric functions 843
 - ORDER BY clause 861
 - PARTITION BY clause 861
 - PERCENT_RANK function 875
 - PERCENTILE_CONT function 888
 - PERCENTILE_DISC function 890
 - range 867
 - RANGE 860
 - RANK function 871
 - ranking functions 843, 860
 - ROLLUP operator 846
 - rows 866
 - ROWS 860
 - semantic phases of execution 844
 - statistical aggregate functions 843
 - statistical functions 860
 - subtotal rows 847
 - using 844
 - window concept 860
 - window framing 860
 - window functions 844
 - window ordering 860
 - window partitioning 860, 861
 - window sizes 860
 - windowing extensions 859
 - windows aggregate functions 843
- OLAP examples 902
 - ascending and descending order for value-based frames 868
 - calculate cumulative sum 904
 - calculate moving average 905
 - computing deltas between adjacent rows 869
 - default window frame for RANGE 909
 - default window frame for ROW 908
 - multiple aggregate functions in a query 906
 - ORDER BY results 905
 - range-based window frames 867
 - row-based window frames 865

Index

- unbounded preceding and unbounded following 908
- unbounded window 868
- using a window with multiple functions 904
- window frame excludes current row 907
- window frame with ROWS vs. RANGE 906
- window functions 870
- window functions in queries 903
- OLAP functions
 - distribution 886
 - inter-row functions 884
 - numerical functions 892
 - ordered sets 886
 - ranking functions 870
 - statistical aggregate 881
 - windowing 859
 - windowing:aggregate functions 880
- OLE DB 320
 - about 319
 - autocommit mode 161
 - connection parameters 327
 - connection pooling 329
 - controlling autocommit behavior 161
 - cursor types 142
 - cursors 158
 - Microsoft Linked Server setup 329
 - ODBC and 319
 - supported interfaces 333
 - supported platforms 320
 - updates 325
 - updating data through a cursor 325
- OLE DB and ADO development
 - about 319
- OLE DB and ADO programming interface
 - about 319
 - introduction 319
- OLE transactions
 - three-tier computing 726
 - three-tier computing terminology 727
- omit_comments a_bit_fielda_translate_log structure [database tools API] 817
- OmniConnect 5
- online analytical processing
 - CUBE operator 855
 - functionality 843
 - NULL values 848
 - ROLLUP operator 846
 - subtotal rows 847
- online backups
 - embedded SQL 467
- Open Client
 - architecture 617
 - autocommit mode 161
 - controlling autocommit behavior 161
 - cursor types 142
 - data type compatibility 619
 - data type ranges 619
 - interface 617
 - introduction 617
 - limitations 623
 - requirements 618
 - SAP Sybase IQ limitations 623
 - SQL 620
 - SQL statements 131
- OPEN statement
 - using cursors in embedded SQL 453
- open_result_set
 - v4 API method 110
- openquery
 - Linked Server 329
- operations a_sql_uint32a_remote_sql structure [database tools API] 782
- optimization
 - defining existing tables and 964
- optimizer estimate
 - a_v4_extfn_estimate 113
- options
 - Open Client 6
 - web services 654
- Oracle data
 - data source names 941
 - environment variables 941
- Oracle data access
 - prerequisites 940
- ORDER BY clause 861, 862
 - sort order 868
- order by list
 - a_v4_extfn_orderby_list 113
- ordered set functions 886
 - PERCENTILE_CONT 886
 - PERCENTILE_DISC 886
- OSGi deployment bundle
 - JDBC 4.0 driver 378
- other_pages a_sql_uint32a_db_info structure [database tools API] 769
- OUT parameters
 - Java in the database 373

- output_dir const char *a_backup_db structure [database tools API] 753
- output_to_file a_bit_fielda_sync_db structure [database tools API] 799
- output_to_mobile_link a_bit_fielda_sync_db structure [database tools API] 799
- output_value
 - a_sqlany_data_valuea_sqlany_bind_param_info structure [SQL Anywhere C API] 529
- OVER clause 860
- overflow errors
 - Open Client data type conversion 619
- OWASP
 - web services 653
- P**
- packages
 - jConnect 382
- Padding() enumeration [database tools API] 749
- page_blocksize a_sql_uint32a_backup_db structure [database tools API] 754
- page_size unsigned shorta_create_db structure [database tools API] 764
- page_size unsigned shorta_sysinfo structure [database tools API] 810
- page_usage a_bit_fielda_db_info structure [database tools API] 769
- parallel backups
 - db_backup function 468
- parameter type
 - a_v4_extfn_describe_parm_type 91
- ParameterName propertySAParameter class [SQL Anywhere .NET API] 292
- parameters
 - substitution 692
- Parameters propertySACommand class [SQL Anywhere .NET API] 237
- partition by
 - column number 114
- PARTITION BY clause 861
- partner certification 1
- Password propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 255
- passwords
 - encrypting in jConnect 382
- patience_retry a_sql_uint32a_remote_sql structure [database tools API] 782
- PERCENT_RANK function 875
- PERCENTILE_CONT function 886, 888
- PERCENTILE_DISC function 886, 890
- performance
 - cursors 153
 - cursors and prefetched rows 154
 - JDBC 395
 - JDBC drivers 378
 - prepared statements 132
- Perl
 - connecting to a database 539
 - DBD::SQLAnywhere 535
 - executing SQL statements 540
 - handling multiple result sets 541
 - inserting rows 542
 - installing Perl/DBI support on Unix 537
 - installing Perl/DBI support on Windows 535
 - writing DBD::SQLAnywhere scripts 538
- Perl DBD::SQLAnywhere
 - about 535
 - introduction to programming 535
- Perl DBI module
 - about 535
- persist_connection a_bit_fielda_sync_db structure [database tools API] 799
- PersistSecurityInfo
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 256
- phases of execution 844
- PHP
 - about 553
 - API reference 566
 - data access 553
 - extension 553
 - running PHP scripts in web pages 554
 - writing scripts 556
- PHP extension
 - introduction to programming 553
 - testing 553
- PHP functions
 - sasql_affected_rows 566
 - sasql_close 567
 - sasql_commit 567
 - sasql_connect 567
 - sasql_data_seek 568
 - sasql_disconnect 568
 - sasql_error 568
 - sasql_errorcode 569
 - sasql_escape_string 569
 - sasql_fetch_array 570

Index

- sasql_fetch_assoc 570
- sasql_fetch_field 571
- sasql_fetch_object 571
- sasql_fetch_row 572
- sasql_field_count 572
- sasql_field_seek 572
- sasql_free_result 573
- sasql_get_client_info 573
- sasql_insert_id 573
- sasql_message 574
- sasql_multi_query 574
- sasql_next_result 575
- sasql_num_fields 575
- sasql_num_rows 575
- sasql_pconnect 576
- sasql_prepare 576
- sasql_query 576
- sasql_real_escape_string 577
- sasql_real_query 577
- sasql_result_all 578
- sasql_rollback 579
- sasql_set_option 579
- sasql_sqlstate 588
- sasql_stmt_affected_rows 580
- sasql_stmt_bind_param 580
- sasql_stmt_bind_param_ex 581
- sasql_stmt_bind_result 582
- sasql_stmt_close 582
- sasql_stmt_data_seek 582
- sasql_stmt_errno 583
- sasql_stmt_error 583
- sasql_stmt_execute 583
- sasql_stmt_fetch 584
- sasql_stmt_field_count 584
- sasql_stmt_free_result 584
- sasql_stmt_insert_id 585
- sasql_stmt_next_result 585
- sasql_stmt_num_rows 586
- sasql_stmt_param_count 586
- sasql_stmt_reset 586
- sasql_stmt_result_metadata 587
- sasql_stmt_send_long_data 587
- sasql_stmt_store_result 587
- sasql_store_result 588
- sasql_use_result 589
- PHP hypertext preprocessor
 - about 553
- physical offset of a window frame 866
- ping a_bit_fielda_sync_db structure [database tools API] 800
- placeholders
 - dynamic SQL 440
- platform certification 3
- platforms
 - cursors 142
- plug-in 11
- pooling
 - connections with .NET Data Provider 169
 - web services 637
- POOLING option
 - .NET Data Provider 169
- Pooling propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 256
- population variance function 882
- positioned DELETE statement
 - about 140
- positioned UPDATE statement
 - about 140
- positioned updates
 - about 138
- POWER function 898
- Precision propertySAPParameter class [SQL Anywhere .NET API] 292
- precision unsigned shorta_sqlany_column_info structure [SQL Anywhere C API] 530
- prefetch
 - cursor performance 153
 - cursors 154
 - fetching multiple rows 139
- prefetch option
 - cursors 154
- PrefetchBuffer
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 256
- PrefetchRows propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 256
- prefixes 845
 - ROLLUP operation 847
 - subtotal rows 847
- preload_dlls char *a_sync_db structure [database tools API] 800
- Prepare method
 - using 133
- PREPARE statement
 - remote data access 954
 - using 440

- PREPARE TRANSACTION statement
 - and Open Client 623
- prepared statements
 - ADO.NET overview 133
 - bind parameters 133
 - cursors 137
 - dropping 133
 - JDBC 395
 - Open Client 621
 - using 132
- PreparedStatement interface
 - about 395
- prepareStatement method
 - JDBC 134
- preparing
 - statements 132
 - to commit 727
- preprocessor
 - about 409
 - running 411
- preserve_identity_values a_bit_fieldan_unload_db
 - structure [database tools API] 832
- preserve_ids a_bit_fieldan_unload_db structure
 - [database tools API] 832
- primary keys
 - retrieving with SACommand 174
 - retrieving with SADDataAdapter 182
- privileges
 - JDBC 400
- ProcedureParameters
 - fieldSAMetaDataCollectionNames class
 - [SQL Anywhere .NET API] 287
- procedures
 - embedded SQL 463
 - requirements for web clients 662
 - result sets in ESQL 464
 - web clients 662
- Procedures fieldSAMetaDataCollectionNames
 - class [SQL Anywhere .NET API] 287
- profiling_uses_single_dbSPACE
 - a_bit_fieldan_unload_db structure
 - [database tools API] 833
- program structure
 - embedded SQL 418
- programming interfaces
 - JDBC API 377
 - Perl DBD::SQLAnywhere API 535
 - Python Database API 545
 - Ruby APIs 591
 - SAP Sybase IQ .NET API 165
 - SAP Sybase IQ embedded SQL 409
 - SAP Sybase IQ OLE DB and ADO APIs 319
 - SAP Sybase IQ PHP DBI 553
 - Sybase Open Client API 617
- progress_index_rtn
 - SET_PROGRESS_CALLBACKa_remot
 - e_sql structure [database tools API] 782
- progress_index_rtn
 - SET_PROGRESS_CALLBACKa_sync_
 - db structure [database tools API] 800
- progress_messages a_bit_fielda_backup_db
 - structure [database tools API] 754
- progress_msg_rtn
 - MSG_CALLBACKa_remote_sql
 - structure [database tools API] 783
- progress_msg_rtn MSG_CALLBACKa_sync_db
 - structure [database tools API] 800
- prompt_again a_bit_fielda_sync_db structure
 - [database tools API] 800
- prompt_for_encrypt_key a_bit_fielda_sync_db
 - structure [database tools API] 800
- properties
 - db_get_property function 476
- protocol_add_cli_bit_to_cli_both
 - a_bit_fielda_sync_db structure [database
 - tools API] 801
- protocol_add_cli_bit_to_cli_max
 - a_bit_fielda_sync_db structure [database
 - tools API] 801
- protocol_add_serv_bit_to_cli_both
 - a_bit_fielda_sync_db structure [database
 - tools API] 801
- protocol_add_serv_bit_to_cli_max
 - a_bit_fielda_sync_db structure [database
 - tools API] 801
- protocol_add_serv_bit_to_serv_both
 - a_bit_fielda_sync_db structure [database
 - tools API] 801
- protocol_add_serv_bit_to_serv_max
 - a_bit_fielda_sync_db structure [database
 - tools API] 801
- protocols
 - configuring web services 627
 - enabling web services 626
- providers
 - supported in .NET 166
- proxy databases 5
- proxy tables 944

Index

pub_name char *a_syncpub structure [database tools API] 809

PUT statement

modifying rows through a cursor 140

multi-row 456

wide 456

Python

closing connections 548

commit method 549

control over type conversion 550

creating connections 548

creating cursors 548

database types 550

executing SQL statements 548

inserting into tables 549

installing Python support on Unix 546

installing Python support on Windows 546

multiple inserts 549

sqlanydb 545

writing sqlanydb scripts 547

Python Database API

introduction to programming 545

Python Database support

about 545

Q

queries

ADO Recordset object 323

ADO Recordset object and cursors 324

prefixes 845

single-row 452

subtotal rows 847

query processing phases

annotation 95

execution 95

optimization 95

plan building 95

query_only a_bit_fielda_change_log structure [database tools API] 758

query_only a_bit_fielda_dblic_info structure [database tools API] 772

queparms char *a_remote_sql structure [database tools API] 783

queparms const char *a_translate_log structure [database tools API] 817

quick start

accessing SAP Sybase IQ web server 660

SAP Sybase IQ web client 658

SAP Sybase IQ web server 625

quiet a_bit_fielda_backup_db structure [database tools API] 754

quiet a_bit_fielda_change_log structure [database tools API] 758

quiet a_bit_fielda_db_info structure [database tools API] 769

quiet a_bit_fielda_dblic_info structure [database tools API] 772

quiet a_bit_fielda_translate_log structure [database tools API] 818

quiet a_bit_fielda_truncate_log structure [database tools API] 821

quiet a_bit_fielda_validate_db structure [database tools API] 823

quiet a_bit_fieldan_erase_db structure [database tools API] 825

quiet a_bit_fieldan_upgrade_db structure [database tools API] 840

quoted identifiers

sql_needs_quotes function 491

QUOTED_IDENTIFIER option

Open Client 6

R

Rails

about 591

installing ActiveRecord adapter 591

range 867

logical offset of a window frame 867

window frame unit 862

window order clause 862

RANGE 860

range specification 864, 867

range-based window frames 867, 868

RANK function 871

rank functions

example 878, 879

ranking functions 843, 860

requirements with OLAP 862

window order clause 862

RAW services

about 629

commenting 634

creating 631

dropping 634

quick start 660

quick start for web clients 658

quick start for web servers 625

- raw_file const char *a_sync_db structure [database tools API] 801
- READ_CLIENT_FILE function
 - ESQL client API callback function 480
- read-only cursors
 - about 142
- receive_a_bit_fielda_remote_sql structure [database tools API] 783
- receive_delay_a_sql_uint32a_remote_sql structure [database tools API] 783
- ReceiveBufferSize propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 313
- recompute_a_bit_fieldan_unload_db structure [database tools API] 833
- record sets
 - ADO programming 325
- RecordsAffected
 - propertySARowUpdatedEventArgs class [SQL Anywhere .NET API] 305
- Recordset ADO object
 - ADO 323
 - ADO programming 326
 - updating data 325
- Recordset object
 - ADO 324
- recovery
 - distributed transactions 729
- recovery_bytes_a_sql_uint32a_translate_log structure [database tools API] 818
- recovery_ops_a_sql_uint32a_translate_log structure [database tools API] 818
- reentrant code
 - multithreaded embedded SQL example 437
- refresh_mat_view_a_bit_fieldan_unload_db structure [database tools API] 833
- registering
 - SAP Sybase IQ .NET Data Provider 195
- reload_connectparms char *an_unload_db structure [database tools API] 833
- reload_db_filename char *an_unload_db structure [database tools API] 833
- reload_db_logname char *an_unload_db structure [database tools API] 834
- reload_filename const char *an_unload_db structure [database tools API] 834
- reload_page_size unsigned shortan_unload_db structure [database tools API] 834
- remote data 943
- remote data access 5, 961
 - internal operations 955
 - remote servers 940
 - troubleshooting 957
- remote procedure calls
 - about 953
- remote servers
 - about 940
 - altering 947
 - creating 940
 - deleting 946
 - external logins 948
 - transaction management 954
- remote_dir const char *an_unload_db structure [database tools API] 834
- remote_output_file_name char *a_remote_sql structure [database tools API] 783
- REMOTEPWD
 - using 384
- remove_encrypted_tables_a_bit_fieldan_unload_db structure [database tools API] 834
- remove_rollback_a_bit_fielda_translate_log structure [database tools API] 818
- removeShutdownHook
 - Java VM shutdown hooks 374
- rename_local_log_a_bit_fielda_backup_db structure [database tools API] 754
- rename_log_a_bit_fielda_backup_db structure [database tools API] 754
- rename_log_a_bit_fielda_remote_sql structure [database tools API] 784
- rename_log_a_bit_fielda_sync_db structure [database tools API] 802
- replace_a_bit_fielda_translate_log structure [database tools API] 818
- replace_db_a_bit_fieldan_unload_db structure [database tools API] 835
- reporting functions 880
 - example 880, 881
- repserver_users const char *a_translate_log structure [database tools API] 819
- request processing
 - embedded SQL 466
- requests
 - aborting 473
- requirements
 - Open Client applications 618
- resend_urgency_a_sql_uint32a_remote_sql structure [database tools API] 784

Index

- reserved `a_bit_fielda_sync_db` structure [database tools API] 802
- reserved `void *a_log_file_info` structure [database tools API] 775
- ReservedWords fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 287
- resource dispensers
 - three-tier computing 727
- resource managers
 - about 725
 - three-tier computing 727
- respect_case `a_bit_fielda_create_db` structure [database tools API] 764
- restart `a_bit_fieldan_upgrade_db` structure [database tools API] 840
- RESTRICT action 968
- Restrictions fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 288
- result sets
 - about ADO Recordset object 323
 - accessing from a web client 677
 - cursors 135
 - Java in the database stored procedures 372
 - JDBC 399
 - metadata 160
 - Open Client 622
 - retrieving from a web service 679
 - stored procedures 464
 - using 137
 - using ADO Recordset object 324
- Results method
 - JDBCExample 399
- retrieving
 - SQLDA 451
- retry_remote_ahead `a_bit_fielda_sync_db` structure [database tools API] 802
- retry_remote_behind `a_bit_fielda_sync_db` structure [database tools API] 802
- RetryConnectionTimeout
 - propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 257
- return codes
 - about 738
- return value
 - describe 93
- return values
 - web clients 677
- rewind
 - v4 API method 123
- ROLLBACK statement
 - cursors 163
- ROLLBACK TO SAVEPOINT statement
 - cursors 164
- RollbackTrans ADO method
 - ADO programming 326
 - updating data 326
- ROLLUP operation 845, 846
 - example 852
 - NULL 848
 - SELECT statement 846
 - subtotal rows 847
- ROLLUP operator 846
- root web services
 - about 635
 - web browsing 654
- row block 116
- row specification 864
- row-based window frames 865
- rows 866
 - physical offset of a window frame 866
 - rows between 1 preceding and 1 following 866
 - rows between 1 preceding and 1 preceding 866
 - rows between current row and current row 866
 - rows between unbounded preceding and current row 866
 - rows between unbounded preceding and unbounded following 866
 - specification 867
 - subtotal rows 847
- ROWS 860
- RowsCopied propertySARowsCopiedEventArgs class [SQL Anywhere .NET API] 307
- RPC option
 - Linked Server 330, 332
- RPC Out option
 - Linked Server 330, 332
- Ruby
 - about 591
 - installing ActiveRecord adapter 591
 - installing native Ruby driver 591
 - installing Ruby/DBI support 592
- Ruby API
 - about 599
 - sqlany_affected_rows function 600
 - sqlany_bind_param function 600

- sqlany_clear_error function 601
 - sqlany_client_version function 601
 - sqlany_commit function 601
 - sqlany_connect function 602
 - sqlany_describe_bind_param function 602
 - sqlany_disconnect function 603
 - sqlany_error function 603
 - sqlany_execute function 604
 - sqlany_execute_direct function 604
 - sqlany_execute_immediate function 605
 - sqlany_fetch_absolute function 605
 - sqlany_fetch_next function 606
 - sqlany_fini function 607
 - sqlany_free_connection function 607
 - sqlany_free_stmt function 608
 - sqlany_get_bind_param_info function 608
 - sqlany_get_column function 609
 - sqlany_get_column_info function 609
 - sqlany_get_next_result function 610
 - sqlany_init function 611
 - sqlany_new_connection function 611
 - sqlany_num_cols function 612
 - sqlany_num_params function 612
 - sqlany_num_rows function 612
 - sqlany_prepare function 613
 - sqlany_rollback function 614
 - sqlany_sqlstate function 614
 - Ruby APIs
 - introduction to programming 591
 - Ruby DBI
 - about 595
 - connection examples 595
 - installing dbd-sqlanywhere 592
 - Ruby on Rails
 - about 591
 - installing ActiveRecord adapter 591
 - RubyGems
 - installing 591
 - runscript a_bit_fieldan_unload_db structure
 - [database tools API] 835
- S**
- sa_set_http_header system procedure
 - example 638
 - SA_TRANSACTION_SNAPSHOT 390
 - SA_TRANSACTION_STATEMENT_READONLY_SNAPSHOT 390
 - SA_TRANSACTION_STATEMENT_SNAPSHOT 390
 - SABulkCopy class [SQL Anywhere .NET API]
 - BatchSize property 214
 - SABulkCopy class [SQL Anywhere .NET API]
 - BulkCopyTimeout property 214
 - SABulkCopy class [SQL Anywhere .NET API]
 - ColumnMappings property 215
 - SABulkCopy class [SQL Anywhere .NET API]
 - description 213
 - SABulkCopy class [SQL Anywhere .NET API]
 - DestinationTableName property 215
 - SABulkCopy class [SQL Anywhere .NET API]
 - NotifyAfter property 216
 - SABulkCopyColumnMapping class [SQL Anywhere .NET API]
 - description 216
 - SABulkCopyColumnMapping class [SQL Anywhere .NET API]
 - DestinationColumn property 217
 - SABulkCopyColumnMapping class [SQL Anywhere .NET API]
 - DestinationOrdinal property 217
 - SABulkCopyColumnMapping class [SQL Anywhere .NET API]
 - SourceColumn property 218
 - SABulkCopyColumnMapping class [SQL Anywhere .NET API]
 - SourceOrdinal property 218
 - SABulkCopyColumnMappingCollection class
 - [SQL Anywhere .NET API] description 219
 - SABulkCopyColumnMappingCollection class
 - [SQL Anywhere .NET API] this property 222
 - SABulkCopyOptions() enumeration [SQL Anywhere .NET API] 212
 - SACAPI_ERROR_SIZE variable [SQL Anywhere C API] 520
 - SACCommand
 - obtaining primary key values 174
 - SACCommand class
 - about 170
 - deleting data 172
 - inserting data 172
 - retrieving data 171
 - updating data 172
 - using prepared statements 133
 - SACCommand class [SQL Anywhere .NET API]
 - CommandText property 234
 - SACCommand class [SQL Anywhere .NET API]
 - CommandTimeout property 235

Index

- SACommand class [SQL Anywhere .NET API]
 - CommandType property 235
- SACommand class [SQL Anywhere .NET API]
 - Connection property 236
- SACommand class [SQL Anywhere .NET API]
 - DbConnection property 236
- SACommand class [SQL Anywhere .NET API]
 - DbParameterCollection property 236
- SACommand class [SQL Anywhere .NET API]
 - DbTransaction property 237
- SACommand class [SQL Anywhere .NET API]
 - description 226
- SACommand class [SQL Anywhere .NET API]
 - DesignTimeVisible property 237
- SACommand class [SQL Anywhere .NET API]
 - Parameters property 237
- SACommand class [SQL Anywhere .NET API]
 - Transaction property 238
- SACommand class [SQL Anywhere .NET API]
 - UpdatedRowSource property 238
- SACommandBuilder class [SQL Anywhere .NET API]
 - DataAdapter property 243
- SACommandBuilder class [SQL Anywhere .NET API]
 - description 239
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
 - All property 224
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
 - ConnectionString property 225
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
 - description 223
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
 - SharedMemory property 225
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
 - TcpOptionsBuilder property 225
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
 - TcpOptionsString property 225
- SAConnection class
 - connecting to a database 168
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - AppInfo property 247
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - AutoStart property 247
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - AutoStop property 247
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - Charset property 247
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - CommBufferSize property 247
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - CommLinks property 248
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - Compress property 248
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - CompressionThreshold property 248
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - ConnectionLifetime property 248
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - ConnectionName property 249
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - ConnectionPool property 249
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - ConnectionReset property 249
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - ConnectionTimeout property 249
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - DatabaseFile property 250
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - DatabaseKey property 250
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - DatabaseName property 250
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - DatabaseSwitches property 250
- SAConnectionStringBuilder class [SQL Anywhere .NET API]
 - DataSourceName property 250

- SACConnectionStringBuilder class [SQL Anywhere .NET API] description 243
- SACConnectionStringBuilder class [SQL Anywhere .NET API] DisableMultiRowFetch property 251
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Elevate property 251
- SACConnectionStringBuilder class [SQL Anywhere .NET API] EncryptedPassword property 251
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Encryption property 251
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Enlist property 252
- SACConnectionStringBuilder class [SQL Anywhere .NET API] FileDataSourceName property 252
- SACConnectionStringBuilder class [SQL Anywhere .NET API] ForceStart property 252
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Host property 252
- SACConnectionStringBuilder class [SQL Anywhere .NET API] IdleTimeout property 252
- SACConnectionStringBuilder class [SQL Anywhere .NET API] InitString property 253
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Integrated property 253
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Kerberos property 253
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Language property 254
- SACConnectionStringBuilder class [SQL Anywhere .NET API] LazyClose property 254
- SACConnectionStringBuilder class [SQL Anywhere .NET API] LivenessTimeout property 254
- SACConnectionStringBuilder class [SQL Anywhere .NET API] LogFile property 254
- SACConnectionStringBuilder class [SQL Anywhere .NET API] MaxPoolSize property 254
- SACConnectionStringBuilder class [SQL Anywhere .NET API] MinPoolSize property 255
- SACConnectionStringBuilder class [SQL Anywhere .NET API] NewPassword property 255
- SACConnectionStringBuilder class [SQL Anywhere .NET API] NodeType property 255
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Password property 255
- SACConnectionStringBuilder class [SQL Anywhere .NET API] PersistSecurityInfo property 256
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Pooling property 256
- SACConnectionStringBuilder class [SQL Anywhere .NET API] PrefetchBuffer property 256
- SACConnectionStringBuilder class [SQL Anywhere .NET API] PrefetchRows property 256
- SACConnectionStringBuilder class [SQL Anywhere .NET API] RetryConnectionTimeout property 257
- SACConnectionStringBuilder class [SQL Anywhere .NET API] ServerName property 257
- SACConnectionStringBuilder class [SQL Anywhere .NET API] StartLine property 257
- SACConnectionStringBuilder class [SQL Anywhere .NET API] Unconditional property 258
- SACConnectionStringBuilder class [SQL Anywhere .NET API] UserID property 258
- SACConnectionStringBuilderBase class [SQL Anywhere .NET API] description 258
- SACConnectionStringBuilderBase class [SQL Anywhere .NET API] Keys property 253, 261, 312

Index

- SACConnectionStringBuilderBase class [SQL Anywhere .NET API] this property 257, 261, 314
- SADDataAdapter
 - obtaining primary key values 182
- SADDataAdapter class
 - about 170
 - deleting data 175
 - inserting data 175
 - retrieving data 177, 178
 - updating data 175
- SADDataAdapter class [SQL Anywhere .NET API] DeleteCommand property 266
- SADDataAdapter class [SQL Anywhere .NET API] description 262
- SADDataAdapter class [SQL Anywhere .NET API] InsertCommand property 267
- SADDataAdapter class [SQL Anywhere .NET API] SelectCommand property 267
- SADDataAdapter class [SQL Anywhere .NET API] TableMappings property 268
- SADDataAdapter class [SQL Anywhere .NET API] UpdateBatchSize property 268
- SADDataAdapter class [SQL Anywhere .NET API] UpdateCommand property 269
- SADDataReader class
 - using 171
- SADDataSourceEnumerator class [SQL Anywhere .NET API] description 271
- SADDataSourceEnumerator class [SQL Anywhere .NET API] Instance property 272
- SADBParametersEditor class [SQL Anywhere .NET API] description 296, 301
- SADDbType propertySAPParameter class [SQL Anywhere .NET API] 292
- SADefault class [SQL Anywhere .NET API] description 272
- SADefault class [SQL Anywhere .NET API] Value field 273
- SAError class [SQL Anywhere .NET API] description 273
- SAError class [SQL Anywhere .NET API] Message property 274
- SAError class [SQL Anywhere .NET API] NativeError property 274
- SAError class [SQL Anywhere .NET API] Source property 274
- SAError class [SQL Anywhere .NET API] SqlState property 274
- SAErrorCollection class [SQL Anywhere .NET API] Count property 276
- SAErrorCollection class [SQL Anywhere .NET API] description 275
- SAErrorCollection class [SQL Anywhere .NET API] this property 276
- SAException class [SQL Anywhere .NET API] description 276
- SAException class [SQL Anywhere .NET API] Errors property 277
- SAException class [SQL Anywhere .NET API] Message property 277
- SAException class [SQL Anywhere .NET API] NativeError property 278
- SAException class [SQL Anywhere .NET API] Source property 278
- SAFactory class [SQL Anywhere .NET API] CanCreateDataSourceEnumerator property 282
- SAFactory class [SQL Anywhere .NET API] description 278
- SAFactory class [SQL Anywhere .NET API] Instance field 282
- SAInfoMessageEventArgs class [SQL Anywhere .NET API] description 282
- SAInfoMessageEventArgs class [SQL Anywhere .NET API] Errors property 283
- SAInfoMessageEventArgs class [SQL Anywhere .NET API] Message property 283
- SAInfoMessageEventArgs class [SQL Anywhere .NET API] MessageType property 283
- SAInfoMessageEventArgs class [SQL Anywhere .NET API] NativeError property 284
- SAInfoMessageEventArgs class [SQL Anywhere .NET API] Source property 284
- SAIsolationLevel propertySATransaction class [SQL Anywhere .NET API] 317
- SAIsolationLevel() enumeration [SQL Anywhere .NET API] 213
- sajdbc4.jar
 - loading SQL Anywhere JDBC driver 387

- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Columns field 284
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] DataSourceInformation field 285
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] DataTypes field 285
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] description 284
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] ForeignKeys field 285
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] IndexColumns field 286
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Indexes field 286
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] MetaDataCollections field 286
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] ProcedureParameters field 287
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Procedures field 287
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] ReservedWords field 287
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Restrictions field 288
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Tables field 288
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] UserDefinedTypes field 288
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Users field 289
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] ViewColumns field 289
- SAMetaDataCollectionNames class [SQL Anywhere .NET API] Views field 289
- sample variance function 882
- samples
 - .NET Data Provider 200
 - DBTools program 736
 - dynamic cursors in embedded SQL 421
 - embedded SQL 420, 422
 - embedded SQL applications 419
 - ODBC 343
 - SimpleViewer 203
 - static cursors in embedded SQL 420
- SAOLEDB
 - OLE DB provider 319
- SAP Sybase IQ .NET API
 - about 165
- SAP Sybase IQ .NET Data Provider
 - about 165
 - samples 200
- SAP Sybase IQ ODBC driver
 - linking on Windows 340
- SAP Sybase IQ Perl DBD::SQLAnywhere DBI module
 - about 535
- SAP Sybase IQ PHP API
 - about 566
- SAP Sybase IQ PHP extension
 - about 553
- SAP Sybase IQ PHP module
 - API reference 566
- SAP Sybase IQ Python Database support
 - about 545
- SAP Sybase IQ Ruby API
 - functions 599
- SAP Sybase IQ web services
 - about 625
- SAPParameter class [SQL Anywhere .NET API] DbType property 290
- SAPParameter class [SQL Anywhere .NET API] description 290
- SAPParameter class [SQL Anywhere .NET API] Direction property 291
- SAPParameter class [SQL Anywhere .NET API] IsNullable property 291
- SAPParameter class [SQL Anywhere .NET API] Offset property 291
- SAPParameter class [SQL Anywhere .NET API] ParameterName property 292
- SAPParameter class [SQL Anywhere .NET API] Precision property 292
- SAPParameter class [SQL Anywhere .NET API] SADbType property 292
- SAPParameter class [SQL Anywhere .NET API] Scale property 293

Index

- SAPParameter class [SQL Anywhere .NET API] Size property 293
- SAPParameter class [SQL Anywhere .NET API] SourceColumn property 294
- SAPParameter class [SQL Anywhere .NET API] SourceColumnNullMapping property 294
- SAPParameter class [SQL Anywhere .NET API] SourceVersion property 294
- SAPParameter class [SQL Anywhere .NET API] Value property 295
- SAPParameterCollection class [SQL Anywhere .NET API] Count property 299
- SAPParameterCollection class [SQL Anywhere .NET API] description 295
- SAPParameterCollection class [SQL Anywhere .NET API] IsFixedSize property 299
- SAPParameterCollection class [SQL Anywhere .NET API] IsReadOnly property 300
- SAPParameterCollection class [SQL Anywhere .NET API] IsSynchronized property 300
- SAPParameterCollection class [SQL Anywhere .NET API] SyncRoot property 300
- SAPParameterCollection class [SQL Anywhere .NET API] this property 300
- SAPPermission class [SQL Anywhere .NET API] description 302
- SAPPermissionAttribute class [SQL Anywhere .NET API] description 303
- SARowsCopiedEventArgs class [SQL Anywhere .NET API] Abort property 307
- SARowsCopiedEventArgs class [SQL Anywhere .NET API] description 306
- SARowsCopiedEventArgs class [SQL Anywhere .NET API] RowsCopied property 307
- SARowUpdatedEventArgs class [SQL Anywhere .NET API] Command property 305
- SARowUpdatedEventArgs class [SQL Anywhere .NET API] description 304
- SARowUpdatedEventArgs class [SQL Anywhere .NET API] RecordsAffected property 305
- SARowUpdatingEventArgs class [SQL Anywhere .NET API] Command property 306
- SARowUpdatingEventArgs class [SQL Anywhere .NET API] description 305
- sasql_affected_rows function (PHP) syntax 566
- sasql_close function (PHP) syntax 567
- sasql_commit function (PHP) syntax 567
- sasql_connect function (PHP) syntax 567
- sasql_data_seek function (PHP) syntax 568
- sasql_disconnect function (PHP) syntax 568
- sasql_error function (PHP) syntax 568
- sasql_errorcode function (PHP) syntax 569
- sasql_escape_string function (PHP) syntax 569
- sasql_fetch_array function (PHP) syntax 570
- sasql_fetch_assoc function (PHP) syntax 570
- sasql_fetch_field function (PHP) syntax 571
- sasql_fetch_object function (PHP) syntax 571
- sasql_fetch_row function (PHP) syntax 572
- sasql_field_count function (PHP) syntax 572
- sasql_field_seek function (PHP) syntax 572
- sasql_free_result function (PHP) syntax 573
- sasql_get_client_info function (PHP) syntax 573
- sasql_insert_id function (PHP) syntax 573
- sasql_message function (PHP) syntax 574

- sasql_multi_query function (PHP)
 - syntax 574
- sasql_next_result function (PHP)
 - syntax 575
- sasql_num_fields function (PHP)
 - syntax 575
- sasql_num_rows function (PHP)
 - syntax 575
- sasql_pconnect function (PHP)
 - syntax 576
- sasql_prepare function (PHP)
 - syntax 576
- sasql_query function (PHP)
 - syntax 576
- sasql_real_escape_string function (PHP)
 - syntax 577
- sasql_real_query function (PHP)
 - syntax 577
- sasql_result_all function (PHP)
 - syntax 578
- sasql_rollback function (PHP)
 - syntax 579
- sasql_set_option function (PHP)
 - syntax 579
- sasql_sqlstate function (PHP)
 - syntax 588
- sasql_stmt_affected_rows function (PHP)
 - syntax 580
- sasql_stmt_bind_param function (PHP)
 - syntax 580
- sasql_stmt_bind_param_ex function (PHP)
 - syntax 581
- sasql_stmt_bind_result function (PHP)
 - syntax 582
- sasql_stmt_close function (PHP)
 - syntax 582
- sasql_stmt_data_seek function (PHP)
 - syntax 582
- sasql_stmt_errno function (PHP)
 - syntax 583
- sasql_stmt_error function (PHP)
 - syntax 583
- sasql_stmt_execute function (PHP)
 - syntax 583
- sasql_stmt_fetch function (PHP)
 - syntax 584
- sasql_stmt_field_count function (PHP)
 - syntax 584
- sasql_stmt_free_result function (PHP)
 - syntax 584
- sasql_stmt_insert_id function (PHP)
 - syntax 585
- sasql_stmt_next_result function (PHP)
 - syntax 585
- sasql_stmt_num_rows function (PHP)
 - syntax 586
- sasql_stmt_param_count function (PHP)
 - syntax 586
- sasql_stmt_reset function (PHP)
 - syntax 586
- sasql_stmt_result_metadata function (PHP)
 - syntax 587
- sasql_stmt_send_long_data function (PHP)
 - syntax 587
- sasql_stmt_store_result function (PHP)
 - syntax 587
- sasql_store_result function (PHP)
 - syntax 588
- sasql_use_result function (PHP)
 - syntax 589
- SATcpOptionsBuilder class [SQL Anywhere .NET API] Broadcast property 311
- SATcpOptionsBuilder class [SQL Anywhere .NET API] BroadcastListener property 311
- SATcpOptionsBuilder class [SQL Anywhere .NET API] ClientPort property 312
- SATcpOptionsBuilder class [SQL Anywhere .NET API] description 308
- SATcpOptionsBuilder class [SQL Anywhere .NET API] DoBroadcast property 312
- SATcpOptionsBuilder class [SQL Anywhere .NET API] Host property 312
- SATcpOptionsBuilder class [SQL Anywhere .NET API] IPV6 property 312
- SATcpOptionsBuilder class [SQL Anywhere .NET API] LDAP property 313
- SATcpOptionsBuilder class [SQL Anywhere .NET API] LocalOnly property 313
- SATcpOptionsBuilder class [SQL Anywhere .NET API] MyIP property 313
- SATcpOptionsBuilder class [SQL Anywhere .NET API] ReceiveBufferSize property 313
- SATcpOptionsBuilder class [SQL Anywhere .NET API] SendBufferSize property 314
- SATcpOptionsBuilder class [SQL Anywhere .NET API] ServerPort property 314

Index

- SATcpOptionsBuilder class [SQL Anywhere .NET API] TDS property 314
- SATcpOptionsBuilder class [SQL Anywhere .NET API] Timeout property 315
- SATcpOptionsBuilder class [SQL Anywhere .NET API] VerifyServerName property 315
- SATransaction class
 - using 186
- SATransaction class [SQL Anywhere .NET API] Connection property 316
- SATransaction class [SQL Anywhere .NET API] DbConnection property 316
- SATransaction class [SQL Anywhere .NET API] description 315
- SATransaction class [SQL Anywhere .NET API] IsolationLevel property 317
- SATransaction class [SQL Anywhere .NET API] SAIsolationLevel property 317
- savepoints
 - cursors 164
- Scale propertySAParameter class [SQL Anywhere .NET API] 293
- scale unsigned shorta_sqlany_column_info structure [SQL Anywhere C API] 531
- scan_log a_bit_fielda_remote_sql structure [database tools API] 784
- schema_reload a_bit_fieldan_unload_db structure [database tools API] 835
- SCROLL cursors
 - embedded SQL 159
 - value-sensitive 151
- scrollable cursors
 - about 139
 - JDBC support 378
- security
 - Java in the database 374
- security context
 - Linked Server 330, 332
- security manager
 - about 374
- SELECT statement
 - single row 452
 - using dynamic SELECT statements 442
- SelectCommand propertySADDataAdapter class [SQL Anywhere .NET API] 267
- semantic phases of execution 844
- send a_bit_fielda_remote_sql structure [database tools API] 784
- send_delay a_sql_uint32a_remote_sql structure [database tools API] 784
- SendBufferSize propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 314
- sensitive cursors
 - about 149
 - cursor properties 142
 - delete example 145
 - embedded SQL 159
 - introduction 144
 - update example 146
- sensitivity
 - cursors 144
 - delete example 145
 - isolation levels 157
 - SAP Sybase IQ cursors 143
 - update example 146
- server 482
- server address
 - embedded SQL function 476
- Server Explorer
 - Visual Studio 203
- server side autocommit
 - about 162
- server_backup a_bit_fielda_backup_db structure [database tools API] 755
- server_backup a_bit_fielda_truncate_log structure [database tools API] 822
- server_mode a_bit_fielda_sync_db structure [database tools API] 802
- server_port a_sql_uint32a_sync_db structure [database tools API] 803
- ServerName propertySAConnectionStringBuilder class [SQL Anywhere .NET API] 257
- ServerPort propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 314
- servers
 - creating 966
 - locating from ESQL 487
 - multiple databases on 9
 - web 625
- services
 - data types 680
 - web 625
- SessionCreateTime property
 - about 649
- SessionID property
 - example 647

- SessionLastTime property
 - about 649
- sessions
 - about 646
 - administering 651
 - creating 647
 - deleting 650
 - detecting 649
 - errors 652
- set_cannot_be_distributed
 - v4 API method 112
- set_error method
 - a_v4_extfn_proc_context 106
- set_generation_number a_bit_fielda_change_log
 - structure [database tools API] 758
- set_value method
 - a_v4_extfn_proc_context 104
- set_window_title_rtn
 - SET_WINDOW_TITLE_CALLBACKa_remote_sql structure [database tools API] 785
- set_window_title_rtn
 - SET_WINDOW_TITLE_CALLBACKa_sync_db structure [database tools API] 803
- setAutoCommit method
 - about 390
- setting
 - values using the SQLDA 449
- setTransactionIsolation method 390
- SharedMemory
 - propertySACommLinksOptionsBuilder class [SQL Anywhere .NET API] 225
- show_undo a_bit_fielda_translate_log structure [database tools API] 819
- simple aggregate functions 859
- SimpleViewer
 - .NET Data Provider sample project 167
 - .NET project 203
- SimpleWin32
 - .NET Data Provider sample project 167
- SimpleXML
 - .NET Data Provider sample project 167
- since_checkpoint a_bit_fielda_translate_log structure [database tools API] 819
- since_time a_sql_uint32a_translate_log structure [database tools API] 819
- site_name const char *an_unload_db structure [database tools API] 835
- Size propertySAPParameter class [SQL Anywhere .NET API] 293
- SMALLDATETIME data type
 - Open client conversion 619
- SMALLMONEY data type
 - Open client conversion 619
- snapshot isolation
 - lost updates 155
 - SQL Anywhere .NET Data Provider 186
- SOAP
 - supplying variables in an envelope 676
- SOAP headers
 - management 668
- SOAP namespace
 - management 673
- SOAP requests
 - structures 693
- SOAP services
 - .NET tutorial 709
 - about 629
 - commenting 634
 - creating 632
 - data types 680
 - dropping 634
 - faults 695
 - JAX-WS tutorial 715
 - SAP Sybase IQ web client tutorial 701
- SOAP system procedures
 - alphabetical list 653
- SOAPHEADER clause
 - managing 668
- software
 - return codes 738
- sort order of ORDER BY in range-based frames 868
- Source propertySAError class [SQL Anywhere .NET API] 274
- Source propertySAException class [SQL Anywhere .NET API] 278
- Source propertySAInfoMessageEventArgs class [SQL Anywhere .NET API] 284
- SourceColumn
 - propertySABulkCopyColumnMapping class [SQL Anywhere .NET API] 218
- SourceColumn propertySAPParameter class [SQL Anywhere .NET API] 294
- SourceColumnNullMapping propertySAPParameter class [SQL Anywhere .NET API] 294

Index

- SourceOrdinal
 - propertySABulkCopyColumnMapping class [SQL Anywhere .NET API] 218
- SourceVersion propertySAParameter class [SQL Anywhere .NET API] 294
- sp_tsql_environment system procedure
 - setting options for jConnect 384
- SQL
 - ADO applications 131
 - applications 131
 - embedded SQL applications 131
 - JDBC applications 131
 - ODBC applications 131
 - Open Client applications 131
- SQL Anywhere .NET API
 - DestinationOrdinalComparer class 219, 222
 - DREnumerator class 270
 - SABulkCopy class 213
 - SABulkCopyColumnMapping class 216
 - SABulkCopyColumnMappingCollection class 219
 - SACCommand class 226
 - SACCommandBuilder class 239
 - SACommLinksOptionsBuilder class 223
 - SAConnectionStringBuilder class 243
 - SAConnectionStringBuilderBase class 258
 - SADDataAdapter class 262
 - SADDataSourceEnumerator class 271
 - SADBParametersEditor class 296, 301
 - SADefault class 272
 - SAError class 273
 - SAErrorCollection class 275
 - SAException class 276
 - SAFactory class 278
 - SAInfoMessageEventArgs class 282
 - SQL Anywhere .NET API
 - SAMetaDataCollectionNames class 284
 - SAParameter class 290
 - SAParameterCollection class 295
 - SAPermission class 302
 - SAPermissionAttribute class 303
 - SARowsCopiedEventArgs class 306
 - SARowUpdatedEventArgs class 304
 - SARowUpdatingEventArgs class 305
 - SATcpOptionsBuilder class 308
 - SATransaction class 315
 - 16 JDBC driver
 - connecting 381
 - URL 381
 - C API a_sqlany_bind_param structure 528
 - C API a_sqlany_bind_param_info structure 528
 - C API a_sqlany_column_info structure 529
 - C API a_sqlany_data_info structure 531
 - C API a_sqlany_data_value structure 532
 - C API SQLAnywhereInterface structure 520
 - JDBC driver
 - about 377
 - choosing a JDBC driver 378
 - loading 4.0 driver 381
 - required files 380
 - using 380
 - applications
 - executing SQL statements 131
 - Communications Area
 - about 435
 - preprocessor utility (iqsqlpp)
 - about 411
 - running 411
 - syntax 411

- SQL statements
 - executing 620
 - web clients 674
 - web services 636
- SQL_ATTR_KEYSET_SIZE
 - ODBC attribute 158
- SQL_ATTR_ROW_ARRAY_SIZE
 - fetching multiple rows 139
 - ODBC attribute 158
- SQL_CALLBACK type declaration
 - about 480
- SQL_CALLBACK_PARM type declaration
 - about 480
- SQL_CURSOR_KEYSET_DRIVEN
 - ODBC cursor attribute 158
- sql_needs_quotes function
 - about 491
- SQL_ROWSET_SIZE
 - fetching multiple rows 139
- SQL/1992
 - SQL preprocessor utility (iqsqlpp) 411
- SQL/1999
 - SQL preprocessor utility (iqsqlpp) 411
- SQL/2003
 - SQL preprocessor utility (iqsqlpp) 411
- SQL/2008
 - SQL preprocessor utility (iqsqlpp) 411
- sqlany_affected_rows function [Ruby API]
 - description 600
- sqlany_affected_rows void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 521
- SQLANY_API_VERSION_1 variable [SQL Anywhere C API] 520
- SQLANY_API_VERSION_2 variable [SQL Anywhere C API] 520
- sqlany_bind_param function [Ruby API]
 - description 600
- sqlany_bind_param void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 521
- sqlany_cancel void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 521
- sqlany_clear_error function [Ruby API]
 - description 601
- sqlany_clear_error void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 521
- sqlany_client_version function [Ruby API]
 - description 601
- sqlany_client_version void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 522
- sqlany_client_version_ex void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 522
- sqlany_commit function [Ruby API]
 - description 601
- sqlany_commit void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 522
- sqlany_connect function [Ruby API]
 - description 602
- sqlany_connect void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 522
- sqlany_describe_bind_param function [Ruby API]
 - description 602
- sqlany_describe_bind_param void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 522
- sqlany_disconnect function [Ruby API]
 - description 603
- sqlany_disconnect void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 522
- sqlany_error function [Ruby API]
 - description 603
- sqlany_error void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 523
- sqlany_execute function [Ruby API]
 - description 604
- sqlany_execute void *SQLAnywhereInterface
 - structure [SQL Anywhere C API] 523
- sqlany_execute_direct function [Ruby API]
 - description 604
- sqlany_execute_direct void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 523
- sqlany_execute_immediate function [Ruby API]
 - description 605
- sqlany_execute_immediate void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 523
- sqlany_fetch_absolute function [Ruby API]
 - description 605
- sqlany_fetch_absolute void
 - *SQLAnywhereInterface structure [SQL Anywhere C API] 523
- sqlany_fetch_next function
 - about 606

Index

- sqlany_fetch_next void *SQLAnywhereInterface structure [SQL Anywhere C API] 523
- sqlany_fini function [Ruby API] description 607
- sqlany_fini void *SQLAnywhereInterface structure [SQL Anywhere C API] 524
- sqlany_fini_ex void *SQLAnywhereInterface structure [SQL Anywhere C API] 524
- sqlany_free_connection function [Ruby API] description 607
- sqlany_free_connection void *SQLAnywhereInterface structure [SQL Anywhere C API] 524
- sqlany_free_stmt function [Ruby API] description 608
- sqlany_free_stmt void *SQLAnywhereInterface structure [SQL Anywhere C API] 524
- sqlany_get_bind_param_info function [Ruby API] description 608
- sqlany_get_bind_param_info void *SQLAnywhereInterface structure [SQL Anywhere C API] 524
- sqlany_get_column function [Ruby API] description 609
- sqlany_get_column void *SQLAnywhereInterface structure [SQL Anywhere C API] 524
- sqlany_get_column_info function [Ruby API] description 609
- sqlany_get_column_info void *SQLAnywhereInterface structure [SQL Anywhere C API] 525
- sqlany_get_data void *SQLAnywhereInterface structure [SQL Anywhere C API] 525
- sqlany_get_data_info void *SQLAnywhereInterface structure [SQL Anywhere C API] 525
- sqlany_get_next_result function [Ruby API] description 610
- sqlany_get_next_result void *SQLAnywhereInterface structure [SQL Anywhere C API] 525
- sqlany_init function [Ruby API] description 611
- sqlany_init void *SQLAnywhereInterface structure [SQL Anywhere C API] 525
- sqlany_init_ex void *SQLAnywhereInterface structure [SQL Anywhere C API] 525
- sqlany_make_connection void *SQLAnywhereInterface structure [SQL Anywhere C API] 526
- sqlany_make_connection_ex void *SQLAnywhereInterface structure [SQL Anywhere C API] 526
- sqlany_new_connection function [Ruby API] description 611
- sqlany_new_connection void *SQLAnywhereInterface structure [SQL Anywhere C API] 526
- sqlany_new_connection_ex void *SQLAnywhereInterface structure [SQL Anywhere C API] 526
- sqlany_num_cols function [Ruby API] description 612
- sqlany_num_cols void *SQLAnywhereInterface structure [SQL Anywhere C API] 526
- sqlany_num_params function [Ruby API] description 612
- sqlany_num_params void *SQLAnywhereInterface structure [SQL Anywhere C API] 526
- sqlany_num_rows function [Ruby API] description 612
- sqlany_num_rows void *SQLAnywhereInterface structure [SQL Anywhere C API] 527
- sqlany_prepare function [Ruby API] description 613
- sqlany_prepare void *SQLAnywhereInterface structure [SQL Anywhere C API] 527
- sqlany_reset void *SQLAnywhereInterface structure [SQL Anywhere C API] 527
- sqlany_rollback function [Ruby API] description 614
- sqlany_rollback void *SQLAnywhereInterface structure [SQL Anywhere C API] 527
- sqlany_send_param_data void *SQLAnywhereInterface structure [SQL Anywhere C API] 527
- sqlany_sqlstate function [Ruby API] description 614
- sqlany_sqlstate void *SQLAnywhereInterface structure [SQL Anywhere C API] 527
- sqlanydb
 - about 545
 - installing on Unix 546
 - installing on Windows 546
 - Python Database API 545
 - writing Python scripts 547

- SQLAnywhereInterface structure [SQL Anywhere C API] description 520
- SQLAnywhereInterface structure [SQL Anywhere C API] dll_handle void * 521
- SQLAnywhereInterface structure [SQL Anywhere C API] initialized int 521
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_affected_rows void * 521
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_bind_param void * 521
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_cancel void * 521
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_clear_error void * 521
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_client_version void * 522
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_client_version_ex void * 522
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_commit void * 522
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_connect void * 522
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_describe_bind_param void * 522
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_disconnect void * 522
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_error void * 523
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_execute void * 523
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_execute_direct void * 523
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_execute_immediate void * 523
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_fetch_absolute void * 523
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_fetch_next void * 523
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_fini void * 524
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_fini_ex void * 524
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_free_connection void * 524
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_free_stmt void * 524
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_get_bind_param_info void * 524
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_get_column void * 524
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_get_column_info void * 525
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_get_data void * 525
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_get_data_info void * 525
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_get_next_result void * 525
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_init void * 525
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_init_ex void * 525
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_make_connection void * 526
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_make_connection_ex void * 526
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_new_connection void * 526
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_new_connection_ex void * 526
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_num_cols void * 526
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_num_params void * 526
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_num_rows void * 527
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_prepare void * 527
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_reset void * 527
- SQLAnywhereInterface structure [SQL Anywhere C API] sqlany_rollback void * 527

Index

- SQLAnywhereInterface structure [SQL Anywhere C API] `sqlany_send_param_data` void * 527
- SQLAnywhereInterface structure [SQL Anywhere C API] `sqlany_sqlstate` void * 527
- SQLBindParameter function
 - ODBC prepared statements 134
- SQLBrowseConnect ODBC function
 - about 346
- SQLBulkOperations
 - ODBC function 140
- SQLCA 435
 - about 435
 - changing 437
 - fields 435
 - managing multiple 439
 - threads 437
- `sqlcabc` SQLCA field 435
- `sqlcaid` SQLCA field 435
- `sqlcode` SQLCA field 435
- SQLConnect ODBC function
 - about 346
- SQLCOUNT 436
- `sqld` SQLDA field
 - about 444
- SQLDA
 - about 443
 - allocating 467
 - descriptors 160
 - dynamic SQL 440
 - fields 444
 - filling using `fill_sqlda` 489
 - filling using `fill_sqlda_ex` 489
 - freeing 488
 - host variables 445
 - `sqlllen` field 446
 - strings and `fill_sqlda` 489
 - strings and `fill_sqlda_ex` 489
- `sqlda_storage` function
 - about 492
- `sqlda_string_length` function
 - about 492
- `sqldabc` SQLDA field
 - about 444
- `sqldaid` SQLDA field
 - about 444
- `sqldata` SQLDA field
 - about 445
- SQLDATETIME data type
 - embedded SQL 427
- `sqldef.h`
 - data types 423
 - software exit codes location 738
- SQLDriverConnect ODBC function
 - about 346
- `sqlerrd` SQLCA field 436
- `sqlerrmc` SQLCA field 435
- `sqlerrml` SQLCA field 435
- `sqlerror` SQLCA field 436, 437
 - elements 435
 - SQLIOCOUNT 436
- `sqlerror` SQLCA field element 436, 437
- `sqlerror_message` function
 - about 493
- `sqlerrp` SQLCA field 436
- SQLExecute function
 - ODBC prepared statements 134
- SQLExtendedFetch
 - fetching multiple rows 139
 - ODBC function 138
- SQLFetch
 - ODBC function 138
- SQLFetchScroll
 - fetching multiple rows 139
 - ODBC function 138
- SQLFreeStmt function
 - ODBC prepared statements 134
- `sqlind` SQLDA field
 - about 445
- SQLIOCOUNT
 - `sqlerror` SQLCA field element 436
- SQLIOESTIMATE 437
- SQLJ standard
 - about 369
- `sqlllen` SQLDA field
 - about 445
 - DESCRIBE statement 446
 - describing values 446
 - retrieving values 451
 - sending values 449
 - values 446
- `sqlname` const char *`a_translate_log` structure [database tools API] 819
- `sqlname` SQLDA field
 - about 445
- `sqlpp` utility
 - supported compilers 415

- SQLPrepare function
 - ODBC prepared statements 134
- SqlState propertySAError class [SQL Anywhere .NET API] 274
- sqlstate SQLCA field 436
- sqltype SQLDA field
 - about 445
 - DESCRIBE statement 446
- sqlvar SQLDA field
 - about 444
 - contents 445
- sqlwarn SQLCA field 436
- SQRT function 899
- square root function 899
- standard deviation
 - functions 881
 - population function 882
 - sample function 882
- standards
 - SQLJ 369
- start_subscriptions a_bit_fieldan_unload_db
 - structure [database tools API] 835
- starting
 - databases using jConnect 384
- startline const char *a_create_db structure [database tools API] 764
- startline const char *an_unload_db structure
 - [database tools API] 836
- StartLine propertySACConnectionStringBuilder
 - class [SQL Anywhere .NET API] 257
- startline_name a_bit_fieldan_unload_db structure
 - [database tools API] 836
- startline_old const char *an_unload_db structure
 - [database tools API] 836
- State property
 - .NET Data Provider 170
- statements
 - insert 132
- static cursors
 - about 148
 - ODBC 158
- static SQL
 - about 440
- statistical aggregate functions 881
- statistical functions 860
 - aggregate 843
- status_rtn STATUS_CALLBACKa_sync_db
 - structure [database tools API] 803
- statusrtn MSG_CALLBACKa_backup_db
 - structure [database tools API] 755
- statusrtn MSG_CALLBACKa_db_info structure
 - [database tools API] 769
- statusrtn MSG_CALLBACKa_translate_log
 - structure [database tools API] 820
- statusrtn MSG_CALLBACKa_validate_db
 - structure [database tools API] 823
- statusrtn MSG_CALLBACKan_unload_db
 - structure [database tools API] 836
- statusrtn MSG_CALLBACKan_upgrade_db
 - structure [database tools API] 840
- STDDEV_POP function 882
- STDDEV_SAMP function 882
- stored procedures
 - creating in embedded SQL 463
 - executing in embedded SQL 463
 - INOUT parameters and Java 373
 - Java in the database 372
 - OUT parameters and Java 373
 - result sets in ESQL 464
 - SAP Sybase IQ .NET Data Provider 185
- strictly_free_memory a_bit_fielda_sync_db
 - structure [database tools API] 803
- strictly_ignore_trigger_ops a_bit_fielda_sync_db
 - structure [database tools API] 803
- string
 - data type 492
- string functions
 - BIT_LENGTH 894
 - LENGTH 898
- strings 424
 - determining length 898
- strings in embedded SQL 424
- structure
 - a_v4_extfn_blob 17
 - a_v4_extfn_blob_istream 21
 - a_v4_extfn_col_subset_of_input 25
 - a_v4_extfn_column_data 22
 - a_v4_extfn_column_list 23
 - a_v4_extfn_estimate 113
 - a_v4_extfn_order_el 24
 - a_v4_extfn_orderby_list 113
 - a_v4_extfn_proc 97
 - a_v4_extfn_proc_context 100
 - a_v4_extfn_table 116
 - a_v4_extfn_table_context 117
 - a_v4_extfn_table_func 124

Index

- structure packing
 - header files 416
- subscriber_username const char *an_unload_db structure [database tools API] 836
- subscription char *a_syncpub structure [database tools API] 809
- subtotal rows 847
 - construction 847
 - definition 846, 855
 - NULL values 848
 - ROLLUP operation 847
- summary information
 - CUBE operator 855
- summary rows
 - ROLLUP operation 847
- supported platforms
 - OLE DB 320
- suppress_statistics a_bit_fieldan_unload_db structure [database tools API] 836
- Sybase IQ ODBC driver
 - driver name 346
- Sybase Open Client support
 - about 617
- sync_opt char *a_sync_db structure [database tools API] 803
- sync_params char *a_sync_db structure [database tools API] 804
- sync_profile char *a_sync_db structure [database tools API] 804
- SyncRoot propertySAPparameterCollection class [SQL Anywhere .NET API] 300
- sys_proc_definer a_bit_fielda_create_db structure [database tools API] 765
- sys_proc_definer unsigned shortan_upgrade_db structure [database tools API] 840
- sysinfo a_sysinfoa_db_info structure [database tools API] 769
- sysinfo a_sysinfoan_unload_db structure [database tools API] 837
- syssservers system table
 - remote servers 940
- system procedures
 - HTTP 653
 - SOAP 653
- system requirements
 - SAP Sybase IQ .NET Data Provider 194
- System.Transactions
 - using 186

T

- table
 - a_v4_extfn_table 116
- table adapter
 - Visual Studio 207
- table constraints 968
- table context
 - a_v4_extfn_table_context 117
 - fetch_block method 121
 - fetch_into method 119
 - rewind method 123
- table functions
 - _close_extfn method 128
 - _fetch_block_extfn method 127
 - _fetch_into_extfn method 126
 - _open_extfn method 126
 - _rewind_extfn method 128
 - a_v4_extfn_table_func 124
- table parameterized function 11, 12
- table UDF 11, 12
- Table UDF 13
- table_id a_sql_uint32a_table_info structure [database tools API] 811
- table_list p_namean_unload_db structure [database tools API] 837
- table_list_provided a_bit_fieldan_unload_db structure [database tools API] 837
- table_name char *a_table_info structure [database tools API] 811
- table_pages a_sql_uint32a_table_info structure [database tools API] 811
- table_used a_sql_uint32a_table_info structure [database tools API] 812
- table_used_pct a_sql_uint32a_table_info structure [database tools API] 812
- TableMappings propertySADDataAdapter class [SQL Anywhere .NET API] 268
- tables
 - creating 968
 - creating proxy 964
 - GLOBAL TEMPORARY 968
 - remote access 919
 - temporary 968
- Tables fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 288
- tables p_namea_validate_db structure [database tools API] 823
- TableViewer
 - .NET Data Provider sample project 167

- TcpOptionsBuilder
 - propertySACommLinksOptionsBuilder class [SQL Anywhere .NET API] 225
- TcpOptionsString
 - propertySACommLinksOptionsBuilder class [SQL Anywhere .NET API] 225
- TDS propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 314
- temp_dir const char *an_unload_db structure [database tools API] 837
- template_name const char *an_unload_db structure [database tools API] 837
- temporary tables 968
 - creating 968
- this
 - propertySABulkCopyColumnMappingCollection class [SQL Anywhere .NET API] 222
- this propertySAConnectionStringBuilderBase class [SQL Anywhere .NET API] 257, 261, 314
- this propertySAErrorCollection class [SQL Anywhere .NET API] 276
- this propertySAParameterCollection class [SQL Anywhere .NET API] 300
- threads
 - Java in the database 372
 - multiple SQLCAs 439
 - multiple thread management in embedded SQL 437
- threads_a_sql_uint32a_remote_sql structure [database tools API] 785
- three-tier computing
 - about 725
 - architecture 725
 - Distributed Transaction Coordinator 727
 - distributed transactions 726
 - EAServer 727
 - Microsoft Transaction Server 727
 - resource dispensers 727
 - resource managers 727
- Time structure
 - time values in .NET Data Provider 184
- timeout callback 481
- Timeout propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 315
- times
 - obtaining with .NET Data Provider 184
- TimeSpan
 - SAP Sybase IQ .NET Data Provider 184
- TIMESTAMP data type
 - Open client conversion 619
- totals_a_table_info *a_db_info structure [database tools API] 770
- TPF 11–13
- tracing
 - .NET support 196
- trans_upload_a_bit_fielda_sync_db structure [database tools API] 804
- transaction isolation level 390
- transaction management 954
- transaction processing
 - using the SAP Sybase IQ .NET Data Provider 186
- Transaction propertySACommand class [SQL Anywhere .NET API] 238
- transaction_logs char *a_remote_sql structure [database tools API] 785
- transactions
 - ADO 326
 - application development 161
 - autocommit mode 161
 - controlling autocommit behavior 161
 - cursors 163
 - distributed 725
 - isolation level 163
 - managing 955
 - OLE DB 326
 - remote data access 954
 - using distributed 728
- TransactionScope class
 - using 186
- triggers_a_bit_fielda_remote_sql structure [database tools API] 785
- troubleshooting
 - cursor positioning 138
 - Java in the database methods 372
 - remote data access 957
- truncate_interrupted chara_truncate_log structure [database tools API] 822
- truncate_log_a_bit_fielda_backup_db structure [database tools API] 755
- truncate_remote_output_file
 - a_bit_fielda_remote_sql structure [database tools API] 785
- truncation
 - FETCH statement 434

Index

- indicator variables 434
 - on FETCH 433
- TSQL_HEX_CONSTANT option
 - Open Client 6
- TSQL_VARIABLES option
 - Open Client 6
- tutorials
 - Create a web server and access it from a web client 697
 - developing a simple .NET database application 203
 - using JAX-WS to access a SOAP/DISH web service 715
 - using SAP Sybase IQ to access a SOAP service 701
 - using the .NET Data Provider Simple code sample 200
 - using the .NET Data Provider Table Viewer code sample 201
 - using Visual C# to access a SOAP/DISH web service 709
- two-phase commit
 - and Open Client 623
 - managing distributed transactions 727
 - three-tier computing 726
- type a_license_typea_dblic_info structure [database tools API] 773
- type a_sqlany_data_typea_sqlany_column_info structure [SQL Anywhere C API] 531
- type a_sqlany_data_typea_sqlany_data_info structure [SQL Anywhere C API] 532
- type a_sqlany_data_typea_sqlany_data_value structure [SQL Anywhere C API] 533
- type chara_validate_db structure [database tools API] 824
- TYPE clause
 - example 675
 - specifying 664
- U**
- UDF 13
- UNBOUNDED FOLLOWING 863, 864
- UNBOUNDED PRECEDING 863
- UNBOUNDED PREDEDING 864
- unbounded window, using 868
- unchained mode
 - controlling 161
 - implementation 162
 - transactions 161
- Unconditional
 - propertySACConnectionStringBuilder class [SQL Anywhere .NET API] 258
- unique
 - constraint 968
- unique cursors
 - about 142
- Unit() enumeration [database tools API] 749
- unload_interrupted charan_unload_db structure [database tools API] 837
- unload_type charan_unload_db structure [database tools API] 838
- Unload() enumeration [database tools API] 749
- unmanaged code
 - dbdata.dll 195
- unordered a_bit_fieldan_unload_db structure [database tools API] 838
- UNSIGNED BIGINT data type
 - embedded SQL 427
- unused a_bit_fielda_remote_sql structure [database tools API] 786
- UPDATE statement
 - JDBC 393
 - positioned 140
- UpdateBatch ADO method
 - ADO programming 326
 - updating data 326
- UpdateBatchSize propertySADDataAdapter class [SQL Anywhere .NET API] 268
- UpdateCommand propertySADDataAdapter class [SQL Anywhere .NET API] 269
- UpdatedRowSource propertySACCommand class [SQL Anywhere .NET API] 238
- updates
 - cursor 325
- upgrade database wizard
 - installing jConnect metadata support 382
- upgrade utility (dbupgrad)
 - installing jConnect metadata support 382
- upld_fail_len a_sql_uint32a_sync_db structure [database tools API] 804
- upload_defs a_syncpub *a_sync_db structure [database tools API] 804
- upload_only a_bit_fielda_sync_db structure [database tools API] 805
- URL clause
 - specifying 663
- URLs
 - database 383

- interpreting 654
- jConnect 383
- session management 648
- SQL Anywhere16 JDBC driver 381
- supplying variables 675
- usage_rtn USAGE_CALLBACKa_sync_db
 - structure [database tools API] 805
- use_fixed_cache a_bit_fielda_sync_db structure
 - [database tools API] 805
- use_hex_offsets a_bit_fielda_remote_sql structure
 - [database tools API] 786
- use_hex_offsets a_bit_fielda_sync_db structure
 - [database tools API] 805
- use_hex_offsets a_bit_fielda_translate_log
 - structure [database tools API] 820
- use_internal_reload a_bit_fieldan_unload_db
 - structure [database tools API] 838
- use_internal_unload a_bit_fieldan_unload_db
 - structure [database tools API] 838
- use_relative_offsets a_bit_fielda_remote_sql
 - structure [database tools API] 786
- use_relative_offsets a_bit_fielda_sync_db structure
 - [database tools API] 805
- use_relative_offsets a_bit_fielda_translate_log
 - structure [database tools API] 820
- used_dialog_allocation a_bit_fielda_sync_db
 - structure [database tools API] 805
- user_name char *a_sync_db structure [database
 - tools API] 806
- User-Agent
 - accessing HTTP headers 641
- UserDefinedTypes
 - fieldSAMetaDataCollectionNames class
 - [SQL Anywhere .NET API] 288
- UserID propertySAConnectionStringBuilder class
 - [SQL Anywhere .NET API] 258
- userlist p_namea_translate_log structure [database
 - tools API] 820
- UserList() enumeration [database tools API] 749
- userlisttype chara_translate_log structure [database
 - tools API] 820
- username char *a_dblic_info structure [database
 - tools API] 773
- Users fieldSAMetaDataCollectionNames class
 - [SQL Anywhere .NET API] 289
- using unbounded windows 868
- utilities
 - return codes 738
 - SQL preprocessor (iqsqlpp) syntax 411

V

- v4 API
 - _close_extfn method 128
 - _fetch_block_extfn method 127
 - _fetch_into_extfn method 126
 - _open_extfn method 126
 - _rewind_extfn method 128
 - alloc method 109
 - close_result_set method 110
 - fetch_block method 121
 - fetch_into method 119
 - get_option method 108
 - open_result_set method 110
 - rewind method 123
 - set_cannot_be_distributed method 112
- valid_data a_bit_fielda_sysinfo structure [database
 - tools API] 810
- Validation() enumeration [database tools API] 750
- value a_sqlany_data_valuea_sqlany_bind_param
 - structure [SQL Anywhere C API] 528
- Value fieldSADefault class [SQL Anywhere .NET
 - API] 273
- Value propertySAParameter class [SQL
 - Anywhere .NET API] 295
- value-based window frames 867
 - ascending and descending order 868
 - ORDER BY clause 868
- value-sensitive cursors
 - about 151
 - delete example 145
 - introduction 144
 - update example 146
- VAR_POP function 882
- VAR_SAMP function 882
- VARCHAR data type
 - embedded SQL 427
- variables
 - accessing in HTTP web services 639
 - in SOAP web services 643
 - supplying to HTTP web services 675
- variance functions 881
- verbose a_bit_fielda_remote_sql structure
 - [database tools API] 786
- verbose a_bit_fielda_sync_db structure [database
 - tools API] 806
- verbose chara_create_db structure [database tools
 - API] 765
- verbose charan_unload_db structure [database tools
 - API] 838

Index

- verbose_download a_bit_fielda_sync_db structure [database tools API] 806
 - verbose_download_data a_bit_fielda_sync_db structure [database tools API] 806
 - verbose_hook a_bit_fielda_sync_db structure [database tools API] 806
 - verbose_minimum a_bit_fielda_sync_db structure [database tools API] 806
 - verbose_msgid a_bit_fielda_sync_db structure [database tools API] 807
 - verbose_option_info a_bit_fielda_sync_db structure [database tools API] 807
 - verbose_protocol a_bit_fielda_sync_db structure [database tools API] 807
 - verbose_row_cnts a_bit_fielda_sync_db structure [database tools API] 807
 - verbose_row_data a_bit_fielda_sync_db structure [database tools API] 807
 - verbose_server a_bit_fielda_sync_db structure [database tools API] 808
 - verbose_upload a_bit_fielda_sync_db structure [database tools API] 808
 - verbose_upload_data a_bit_fielda_sync_db structure [database tools API] 808
 - Verbosity() enumeration [database tools API] 750
 - VerifyServerName propertySATcpOptionsBuilder class [SQL Anywhere .NET API] 315
 - version unsigned shorta_backup_db structure [database tools API] 755
 - version unsigned shorta_change_log structure [database tools API] 759
 - version unsigned shorta_create_db structure [database tools API] 765
 - version unsigned shorta_db_info structure [database tools API] 770
 - version unsigned shorta_db_version_info structure [database tools API] 771
 - version unsigned shorta_dblic_info structure [database tools API] 773
 - version unsigned shorta_log_file_info structure [database tools API] 775
 - version unsigned shorta_remote_sql structure [database tools API] 786
 - version unsigned shorta_sync_db structure [database tools API] 808
 - version unsigned shorta_translate_log structure [database tools API] 820
 - version unsigned shorta_truncate_log structure [database tools API] 822
 - version unsigned shorta_validate_db structure [database tools API] 824
 - version unsigned shortan_erase_db structure [database tools API] 825
 - version unsigned shortan_unload_db structure [database tools API] 839
 - version unsigned shortan_upgrade_db structure [database tools API] 841
 - Version() enumeration [database tools API] 750
 - ViewColumns fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 289
 - Views fieldSAMetaDataCollectionNames class [SQL Anywhere .NET API] 289
 - visible changes
 - cursors 144
 - Visual Basic
 - support in .NET Data Provider 165
 - tutorial 203
 - Visual C#
 - tutorial 203
 - Visual C++
 - embedded SQL support 415
 - VM
 - Java VM 370
 - shutdown hooks 374
 - starting Java 374
 - stopping Java 374
- ## W
- wait_after_end a_bit_fielda_backup_db structure [database tools API] 755
 - wait_before_start a_bit_fielda_backup_db structure [database tools API] 756
 - warningrtn MSG_CALLBACKa_remote_sql structure [database tools API] 787
 - warningrtn MSG_CALLBACKa_sync_db structure [database tools API] 808
 - web clients
 - about 658
 - accessing result sets 677
 - function and procedure requirements 662
 - functions and procedures 662
 - managing HTTP request headers 666
 - managing SOAP headers 668
 - managing SOAP namespace 673
 - quick start 658
 - retrieving result sets 679
 - specifying ports 666
 - specifying request types 664

- SQL statements 674
- substitution parameters 692
- supplying variables 675
- URL clause 663
- web pages
 - customizing 638
 - running PHP scripts in 554
- web servers
 - about 625
 - application development 638
 - configuring protocols 627
 - enabling protocols 626
 - PHP API 553
 - quick start 625
 - starting multiple 628
- web services
 - about 625
 - accessing 658
 - accessing HTTP variables and headers 639
 - accessing result sets 677
 - accessing SOAP headers 643
 - alphabetical list of system procedures 653
 - altering 631
 - array types 687
 - character sets 652
 - client log file, about 694
 - commenting 634
 - configuring protocols 627
 - connection properties 654
 - creating 631
 - cross site scripting 653
 - data types 680
 - developing 638
 - dropping 634
 - enabling protocols 626
 - errors 695
 - host variables 640
 - HTTP_HEADER example 640
 - HTTP_VARIABLE example 640
 - interpreting URLs 654
 - list of web services-related system procedures 653
 - logging 694
 - maintaining 631
 - managing 629
 - managing sessions 646
 - MIME types tutorial 697
 - NEXT_HTTP_HEADER example 640
 - NEXT_HTTP_VARIABLE example 640
 - options 654
 - pooling 637
 - references 695
 - retrieving result sets 679
 - root 635
 - SOAP data types 686
 - SOAP/DISH tutorial 701
 - SQL statements 636
 - structure types 687
 - types 629
- WebClientLogFile property
 - server option 694
- WebClientLogging property
 - server option 694
- wide fetches
 - about 139, 456
 - ESQL 456
- wide inserts
 - ESQL 456
 - JDBC 398
- wide puts
 - ESQL 456
- WIDTH_BUCKET function 899
- window
 - frame clause 862
 - operator 859
 - order clause 861, 862
 - ordering 860, 861
- window frame unit 862, 866, 867
 - range 867
 - rows 866
- window frames 860, 862
 - range based 867, 868
 - row based 865
- window functions
 - aggregate 843, 860
 - distribution 860
 - framing 862
 - ordering 861
 - OVER clause 860
 - partitioning 861
 - ranking 860
 - statistical 860
 - window function type 859
 - window name or specification 859
 - window partition 859
- window partitioning 860, 861
 - clause 861
 - GROUP BY operator 861

Index

- window sizes
 - RANGE 860
 - ROWS 860
- windowing
 - aggregate functions 860, 880
 - extensions 859
 - functions 860
 - partitions 859
- Windows
 - OLE DB support 319
- Windows Mobile
 - OLE DB support 319
- WITH HOLD clause
 - cursors 138
- work tables
 - cursor performance 153
- WRITE_CLIENT_FILE function
 - ESQL client API callback function 480
- wsimport
 - JAX-WS and web services 719

X

- XML services
 - about 629

- commenting 634
- creating 631
- dropping 634
- quick start 660
- quick start for web clients 658
- quick start for web servers 625
- XMLCONCAT function
 - example 639
- XMLELEMENT function
 - example 639
- XSS
 - web services 653

Z

- zap_current_offset char *a_change_log structure
 - [database tools API] 759
- zap_starting_offset char *a_change_log structure
 - [database tools API] 759