



Server-Library/C リファレンス・マニュアル

## **Open Server™**

15.7

ドキュメント ID : DC32626-01-1570-01

改訂 : 2012 年 6 月

Copyright © 2012 by Sybase, Inc. All rights reserved.

このマニュアルは Sybase ソフトウェアの付属マニュアルであり、新しいマニュアルまたはテクニカル・ノートで特に示されないかぎり、後続のリリースにも付属します。このマニュアルの内容は予告なしに変更されることがあります。このマニュアルに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、無断で使用することはできません。

このマニュアルの内容を弊社の書面による事前許可を得ずに、電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても、複製、転載、翻訳することを禁じます。

Sybase の商標は、the Sybase trademarks page (<http://www.sybase.com/detail?id=1011207>) で確認できます。Sybase およびこのリストに掲載されている商標は、米国法人 Sybase, Inc. の商標です。® は、米国における登録商標であることを示します。

このマニュアルに記載されている SAP、その他の SAP 製品、サービス、および関連するロゴは、ドイツおよびその他の国における SAP AG の商標または登録商標です。

Java および Java 関連の商標は、米国およびその他の国における Oracle およびその関連会社の商標または登録商標です。

Unicode と Unicode のロゴは、Unicode, Inc. の登録商標です。

このマニュアルに記載されている上記以外の社名および製品名は、当該各社の商標または登録商標の場合があります。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# 目次

はじめに.....	xix
<b>第 1 章</b>	<b>Open Server の概要..... 1</b>
	クライアント／サーバ・アーキテクチャの概要..... 1
	クライアントのタイプ..... 2
	サーバのタイプ..... 2
	Open Server の構成..... 4
	スタンドアロン Open Server アプリケーション..... 4
	補助 Open Server アプリケーション..... 4
	ゲートウェイ Open Server アプリケーション..... 5
	Open Server..... 6
	Open Server ライブラリ..... 6
	ネットワーク・サービス..... 7
	Open Server の使用..... 8
	CS_CONTEXT 構造体..... 8
	簡単なプログラムの手順..... 9
	基本的な Open Server プログラム..... 10
	Open Server イベント..... 14
	デフォルトのイベント・ハンドラ..... 15
	クライアントでは開始されないイベント..... 15
	レジスタード・プロシージャ..... 16
	クライアントへの結果の送信..... 16
	結果データの種類..... 16
	結果の順序..... 18
	エラー処理..... 18
	マルチスレッド・プログラミング..... 18
<b>第 2 章</b>	<b>トピック名..... 19</b>
	アテンション・イベント..... 20
	割り込みレベルのアクティビティ..... 20
	アテンション・イベントに関するコーディングの考慮事項... 21
	切断の処理..... 22
	例..... 23

ブラウズ・モード .....	23
例 .....	25
機能 .....	25
要求機能 .....	25
応答機能 .....	28
透過的ネゴシエーション .....	30
サーバワイドのデフォルト .....	31
明示的ネゴシエーション .....	35
機能情報の特定の検索 .....	37
バージョン 10.0 以前のクライアントに関する注意 .....	38
例 .....	38
クライアント・コマンド・エラー .....	38
srv_sendinfo .....	38
長いメッセージの連続化 .....	39
拡張エラー・データ .....	40
接続マイグレーション .....	41
バッチ内マイグレーションとアイドル・ マイグレーション .....	41
コンテキスト・マイグレーション .....	42
接続マイグレーションで使用される API .....	43
クライアントに対する異なるサーバへのマイグレー ションの指示 .....	50
マイグレートされたクライアントからの接続の受け入れ .....	54
エラー・メッセージ .....	55
CS_BROWSEDESC 構造体 .....	55
CS_DATAFMT 構造体 .....	57
CS_IODESC 構造体 .....	61
CS-Library .....	62
共通ルーチン .....	62
共通データ 構造体 .....	63
エラー処理 .....	64
CS_SERVERMSG 構造体 .....	64
カーソル .....	66
カーソルの概要 .....	67
カーソルの利点 .....	67
Open Server アプリケーションとカーソル .....	68
カーソル要求の処理 .....	77
キー・データ .....	82
更新カラム .....	82
例 .....	83
スクロール可能カーソル .....	83
SRV_CURDESC2 構造体 .....	83
データ・ストリーム・メッセージ .....	86
データ・ストリーム・メッセージとは .....	86
クライアント・データ・ストリーム・メッセージの取得 .....	87

クライアントへのデータ・ストリーム・メッセージの送信 ...	87
ディレクトリ・サービス .....	88
ディレクトリ・ドライバの指定 .....	88
ディレクトリへの Open Server アプリケーションの登録 .....	89
動的 SQL .....	90
動的 SQL の利点 .....	90
動的 SQL 要求の処理 .....	91
例 .....	95
動的なリスナ .....	95
設定 .....	95
動的なリスナの起動 .....	96
エラー .....	97
エラーのタイプ .....	97
エラーの重大度 .....	98
エラー番号と対応するメッセージ・テキスト .....	100
例 .....	100
イベント .....	100
イベントとは .....	100
イベント・ハンドラとは .....	101
標準イベント .....	102
プログラマ定義イベント .....	108
例 .....	109
ゲートウェイ・アプリケーション .....	109
パススルー・モード .....	111
国際化サポート .....	111
Open Server アプリケーションのローカライゼーション .....	112
ローカライズされたクライアントのサポート .....	113
CS_LOCALE 構造体を使用したカスタム・	
ローカライゼーション値の設定 .....	114
クライアント要求に対する応答 .....	117
ローカライゼーション・プロパティ .....	119
ローカライゼーションの例 .....	120
言語呼び出し .....	120
ログイン・リダイレクトと拡張 HA フェールオー	
バのサポート .....	121
メッセージ .....	122
マルチスレッド・プログラミング .....	123
スレッドについて .....	123
スレッドのタイプ .....	123
スケジューリング .....	127
ツールと手法 .....	130
プログラミングに関する注意事項 .....	133
例 .....	134
ネゴシエートされた動作 .....	135

ログイン・ネゴシエーション.....	135
アドホック・ネゴシエーション.....	137
例.....	138
オプション.....	138
SRV_OPTION イベント・ハンドラの内部構造.....	138
オプションの説明とデフォルト値.....	139
例.....	143
パラメータ.....	143
バッチ・パラメータ.....	143
部分更新.....	144
Open Server の設定.....	144
パススルー・モード.....	146
通常パススルー・モード.....	147
イベント・ハンドラ・パススルー・モード.....	149
パラメータとロー・データの処理.....	152
用語についての注意.....	152
Open Server のデータ処理モデル.....	152
パラメータの取得.....	153
ローの返送.....	154
リターン・パラメータの返送.....	154
記述、バインド、転送.....	155
言語データ・ストリームでのパラメータの返送.....	157
例.....	157
プロパティ.....	157
コンテキスト・プロパティ.....	158
サーバ・プロパティ.....	159
スレッド・プロパティ.....	168
レジスタード・プロシージャ.....	185
標準リモート・プロシージャ・コール.....	185
レジスタード・プロシージャの利点.....	185
ノーティフィケーション・プロシージャ.....	186
レジスタード・プロシージャの作成.....	187
レジスタード・プロシージャのメカニズム.....	187
システム・レジスタード・プロシージャ.....	189
レジスタード・プロシージャでのコールバック・ ハンドラの使用.....	190
例.....	192
リモート・プロシージャ・コール.....	192
例.....	193
ラージ・オブジェクト・パラメータの取得.....	194
セキュリティ・サービス.....	196
セキュリティ・サービス・プロパティ.....	197
Open Server でのセキュリティ・サービスの機能.....	207
Open Server アプリケーションでのセキュリティ・ メカニズムの使用.....	208

	アクティブなセキュリティ・サービスの決定 .....	212
	Open Server アプリケーションでのセキュリティ・ サービスの使用例 .....	213
	text と image .....	225
	text および image データの処理 .....	225
	例 .....	228
	データ型 .....	229
	データ型を操作するルーチン .....	231
	Open Server のデータ型 .....	231
<b>第 3 章</b>	<b>ルーチン .....</b>	<b>243</b>
	srv_alloc .....	248
	srv_alt_bind .....	250
	srv_alt_descfmt .....	255
	srv_alt_header .....	259
	srv_alt_xferdata .....	261
	srv_bind .....	263
	srv_bmove .....	269
	srv_bzero .....	271
	srv_callback .....	272
	srv_capability .....	276
	srv_capability_info .....	278
	srv_createmsgq .....	282
	srv_createmutex .....	284
	srv_createproc .....	286
	srv_cursor_props .....	288
	srv_cursor_props2 .....	291
	srv_dbg_stack .....	292
	srv_dbg_switch .....	295
	srv_define_event .....	295
	srv_deletemsgq .....	298
	srv_deletemutex .....	300
	srv_descfmt .....	302
	srv_dynamic .....	306
	srv_envchange .....	310
	srv_event .....	313
	srv_event_deferred .....	315
	srv_free .....	318
	srv_freeserveraddr .....	319
	srv_get_data .....	320
	srv_get_text .....	321
	srv_getloginfo .....	324
	srv_getmsgq .....	326
	srv_getobjid .....	329

---

srv_getobjname.....	331
srv_getserverbyname.....	334
srv_handle.....	335
srv_init.....	338
srv_langcpy.....	340
srv_langlen.....	343
srv_lockmutex.....	344
srv_log.....	347
srv_mask.....	349
srv_msg.....	351
srv_negotiate.....	355
srv_numparams.....	363
srv_options.....	365
srv_orderby.....	372
srv_poll (UNIX のみ).....	374
srv_props.....	377
srv_putmsgq.....	385
srv_realloc.....	387
srv_recvpassthru.....	389
srv_regcreate.....	391
srv_regdefine.....	393
srv_regdrop.....	396
srv_regexec.....	398
srv_reginit.....	400
srv_reglist.....	403
srv_reglistfree.....	405
srv_regnowatch.....	406
srv_regparam.....	408
srv_regwatch.....	411
srv_regwatchlist.....	415
srv_rpcdb.....	416
srv_rpcname.....	418
srv_rpcnumber.....	421
srv_rpcoptions.....	423
srv_rpcowner.....	424
srv_run.....	426
srv_s_ssl_local_id.....	427
srv_select (UNIX のみ).....	428
srv_send_ctlinfo.....	431
srv_send_data.....	433
srv_send_text.....	437
srv_senddone.....	440
srv_sendinfo.....	445
srv_sendpassthru.....	449
srv_sendstatus.....	452



---

	srv_setcoltype .....	453
	srv_setcontrol .....	455
	srv_setloginfo .....	458
	srv_setpri .....	460
	srv_signal (UNIX のみ) .....	462
	srv_sleep .....	464
	srv_spawn .....	468
	srv_symbol .....	471
	srv_tabcolname .....	476
	srv_tabname .....	478
	srv_termproc .....	480
	srv_text_info .....	482
	srv_thread_props .....	485
	srv_timedsleep .....	490
	srv_ucwakeup .....	492
	srv_unlockmutex .....	494
	srv_version .....	495
	srv_wakeup .....	497
	srv_xferdata .....	500
	srv_yield .....	503
<b>第 4 章</b>	<b>システム・レジスタード・プロシージャ .....</b>	<b>505</b>
	sp_ps .....	505
	sp_regcreate .....	508
	sp_regdrop .....	515
	sp_reglst .....	516
	sp_regnowatch .....	517
	sp_regwatch .....	518
	sp_regwatchlist .....	520
	sp_serverinfo .....	520
	sp_terminate .....	521
	sp_who .....	523
<b>付録 A</b>	<b>Open Server エラー・メッセージ .....</b>	<b>527</b>
	エラー 16000 .....	527
	エラー 16001 .....	527
	エラー 16002 ( 廃止予定 ) .....	528
	エラー 16003 .....	528
	エラー 16004 .....	528
	エラー 16005 .....	529
	エラー 16006 ( 廃止予定 ) .....	529
	エラー 16007 ( 廃止予定 ) .....	529
	エラー 16008 .....	530
	エラー 16009 .....	530

エラー 16010 .....	530
エラー 16011 ( 廃止予定 ) .....	531
エラー 16012 .....	531
エラー 16013 .....	531
エラー 16014 ( 廃止予定 ) .....	532
エラー 16015 ( 廃止予定 ) .....	532
エラー 16016 .....	532
エラー 16017 ( 廃止予定 ) .....	533
エラー 16018 ( 廃止予定 ) .....	533
エラー 16019 .....	533
エラー 16020 ( 廃止予定 ) .....	534
エラー 16021 ( 廃止予定 ) .....	534
エラー 16022 ( 廃止予定 ) .....	535
エラー 16023 ( 廃止予定 ) .....	535
エラー 16024 ( 廃止予定 ) .....	535
エラー 16025 ( 廃止予定 ) .....	535
エラー 16026 ( 廃止予定 ) .....	536
エラー 16027 ( 廃止予定 ) .....	536
エラー 16028 ( 廃止予定 ) .....	536
エラー 16029 .....	537
エラー 16030 ( 廃止予定 ) .....	537
エラー 16031 ( 廃止予定 ) .....	537
エラー 16032 .....	538
エラー 16033 .....	538
エラー 16034 .....	538
エラー 16035 ( 廃止予定 ) .....	539
エラー 16036 ( 廃止予定 ) .....	539
エラー 16037 ( 廃止予定 ) .....	539
エラー 16038 ( 廃止予定 ) .....	540
エラー 16039 ( 廃止予定 ) .....	540
エラー 16040 ( 廃止予定 ) .....	540
エラー 16041 ( 廃止予定 ) .....	541
エラー 16042 ( 廃止予定 ) .....	541
エラー 16043 .....	541
エラー 16044 .....	542
エラー 16045 .....	542
エラー 16046 .....	542
エラー 16047 .....	543
エラー 16048 .....	543
エラー 16049 .....	544
エラー 16050 .....	544
エラー 16051 ( 廃止予定 ) .....	544
エラー 16052 .....	545
エラー 16053 .....	545
エラー 16054 .....	545

エラー 16055 .....	546
エラー 16056 .....	546
エラー 16057 .....	546
エラー 16058 .....	547
エラー 16059 (廃止予定).....	547
エラー 16060 (廃止予定).....	547
エラー 16061 .....	548
エラー 16062 .....	548
エラー 16063 .....	548
エラー 16064 (廃止予定).....	549
エラー 16065 .....	549
エラー 16066 (廃止予定).....	549
エラー 16067 .....	550
エラー 16068 (廃止予定).....	550
エラー 16069 (廃止予定).....	550
エラー 16070 (廃止予定).....	551
エラー 16071 (廃止予定).....	551
エラー 16072 .....	551
エラー 16073 .....	552
エラー 16074 .....	552
エラー 16075 .....	552
エラー 16076 .....	553
エラー 16077 (廃止予定).....	553
エラー 16078 (廃止予定).....	553
エラー 16079 (廃止予定).....	554
エラー 16080 .....	554
エラー 16081 (廃止予定).....	554
エラー 16082 (廃止予定).....	555
エラー 16083 (廃止予定).....	555
エラー 16084 (廃止予定).....	555
エラー 16085 (廃止予定).....	556
エラー 16086 (廃止予定).....	556
エラー 16087 (廃止予定).....	556
エラー 16088 (廃止予定).....	557
エラー 16089 (廃止予定).....	557
エラー 16090 (廃止予定).....	557
エラー 16091 (廃止予定).....	558
エラー 16092 (廃止予定).....	558
エラー 16093 (廃止予定).....	558
エラー 16094 .....	558
エラー 16095 (廃止予定).....	559
エラー 16096 (廃止予定).....	559
エラー 16097 .....	559
エラー 16098 (廃止予定).....	560
エラー 16099 (廃止予定).....	560

エラー 16100 .....	560
エラー 16101 ( 廃止予定 ) .....	561
エラー 16102 ( 廃止予定 ) .....	561
エラー 16103 .....	561
エラー 16104 .....	562
エラー 16105 .....	562
エラー 16106 .....	562
エラー 16107 ( 廃止予定 ) .....	563
エラー 16108 .....	563
エラー 16109 .....	563
エラー 16110 .....	564
エラー 16111 .....	564
エラー 16112 .....	565
エラー 16113 .....	565
エラー 16114 ( 廃止予定 ) .....	565
エラー 16115 .....	566
エラー 16116 ( 廃止予定 ) .....	566
エラー 16117 ( 廃止予定 ) .....	566
エラー 16118 .....	567
エラー 16119 .....	567
エラー 16120 .....	567
エラー 16121 .....	568
エラー 16122 .....	568
エラー 16123 .....	569
エラー 16124 .....	569
エラー 16125 .....	569
エラー 16126 .....	570
エラー 16127 .....	570
エラー 16128 .....	570
エラー 16129 .....	571
エラー 16130 .....	571
エラー 16131 .....	572
エラー 16132 .....	572
エラー 16133 .....	572
エラー 16134 .....	573
エラー 16135 .....	573
エラー 16136 .....	574
エラー 16137 .....	574
エラー 16138 .....	575
エラー 16139 ( 廃止予定 ) .....	575
エラー 16140 .....	575
エラー 16141 .....	576
エラー 16142 .....	576
エラー 16143 .....	576
エラー 16144 .....	577

エラー 16145 .....	577
エラー 16146 (廃止予定).....	578
エラー 16147 .....	578
エラー 16148 .....	578
エラー 16149 .....	579
エラー 16150 .....	579
エラー 16151 .....	579
エラー 16152 (廃止予定).....	580
エラー 16153 (廃止予定).....	580
エラー 16154 (廃止予定).....	580
エラー 16155 (廃止予定).....	580
エラー 16156 .....	581
エラー 16157 .....	581
エラー 16158 .....	581
エラー 16159 .....	582
エラー 16160 .....	582
エラー 16161 (廃止予定).....	582
エラー 16162 .....	583
エラー 16163 .....	583
エラー 16164 (廃止予定).....	583
エラー 16165 .....	584
エラー 16166 .....	584
エラー 16167 .....	584
エラー 16168 .....	585
エラー 16169 .....	585
エラー 16170 .....	585
エラー 16171 .....	586
エラー 16172 .....	586
エラー 16173 (廃止予定).....	586
エラー 16174 .....	587
エラー 16175 .....	587
エラー 16176 .....	587
エラー 16177 .....	588
エラー 16178 .....	588
エラー 16179 .....	588
エラー 16180 .....	589
エラー 16181 .....	589
エラー 16182 .....	589
エラー 16183 (廃止予定).....	590
エラー 16184 (廃止予定).....	590
エラー 16185 (廃止予定).....	590
エラー 16186 (廃止予定).....	590
エラー 16187 (廃止予定).....	591
エラー 16188 .....	591
エラー 16189 (廃止予定).....	591

エラー 16190 ( 廃止予定 )	592
エラー 16191 ( 廃止予定 )	592
エラー 16192	592
エラー 16193 ( 廃止予定 )	593
エラー 16194 ( 廃止予定 )	593
エラー 16195	594
エラー 16196 ( 廃止予定 )	594
エラー 16197 ( 廃止予定 )	594
エラー 16198	594
エラー 16199 ( 廃止予定 )	595
エラー 16200	595
エラー 16201	596
エラー 16202	596
エラー 16203	596
エラー 16204	597
エラー 16205 ( 廃止予定 )	597
エラー 16206 ( 廃止予定 )	597
エラー 16207	598
エラー 16208	598
エラー 16209 ( 廃止予定 )	598
エラー 16210	599
エラー 16211 ( 廃止予定 )	599
エラー 16212	599
エラー 16213	600
エラー 16214 ( 廃止予定 )	600
エラー 16215 ( 廃止予定 )	600
エラー 16216	601
エラー 16217	601
エラー 16218	601
エラー 16219	602
エラー 16220	602
エラー 16221	603
エラー 16222	603
エラー 16223	603
エラー 16224 ( 廃止予定 )	604
エラー 16225 ( 廃止予定 )	604
エラー 16226 ( 廃止予定 )	604
エラー 16227	605
エラー 16228 ( 廃止予定 )	605
エラー 16229	605
エラー 16230	606
エラー 16231	606
エラー 16232	606
エラー 16233	607
エラー 16234	607

エラー 16235 .....	607
エラー 16236 .....	608
エラー 16237 .....	608
エラー 16238 .....	608
エラー 16239 (廃止予定).....	609
エラー 16240 .....	609
エラー 16241 .....	610
エラー 16242 (廃止予定).....	610
エラー 16243 (廃止予定).....	610
エラー 16244 .....	611
エラー 16245 (廃止予定).....	611
エラー 16246 (廃止予定).....	611
エラー 16247 .....	612
エラー 16248 .....	612
エラー 16249 (廃止予定).....	612
エラー 16250 (廃止予定).....	613
エラー 16251 .....	613
エラー 16252 .....	613
エラー 16253 .....	614
エラー 16254 .....	614
エラー 16255 .....	614
エラー 16256 .....	615
エラー 16257 .....	615
エラー 16258 (廃止予定).....	616
エラー 16259 (廃止予定).....	616
エラー 16260 .....	616
エラー 16261 .....	617
エラー 16262 .....	617
エラー 16263 .....	617
エラー 16264 (廃止予定).....	618
エラー 16265 .....	618
エラー 16266 .....	618
エラー 16267 .....	619
エラー 16268 (廃止予定).....	619
エラー 16269 .....	619
エラー 16270 (廃止予定).....	620
エラー 16271 .....	620
エラー 16272 .....	621
エラー 16273 .....	621
エラー 16274 .....	621
エラー 16275 (廃止予定).....	622
エラー 16276 .....	622
エラー 16277 (廃止予定).....	622
エラー 16278 .....	623
エラー 16279 .....	623

エラー 16280 .....	623
エラー 16281 .....	624
エラー 16282 .....	624
エラー 16283 .....	624
エラー 16284 .....	625
エラー 16285 .....	625
エラー 16286 .....	626
エラー 16287 ( 廃止予定 ) .....	626
エラー 16288 ( 廃止予定 ) .....	626
エラー 16289 .....	627
エラー 16290 ( 廃止予定 ) .....	627
エラー 16291 .....	627
エラー 16292 ( 廃止予定 ) .....	628
エラー 16293 .....	628
エラー 16294 ( 廃止予定 ) .....	628
エラー 16295 .....	629
エラー 16296 .....	629
エラー 16297 .....	630
エラー 16298 .....	630
エラー 16299 ( 廃止予定 ) .....	630
エラー 16300 .....	631
エラー 16301 .....	631
エラー 16302 .....	631
エラー 16303 .....	632
エラー 16304 .....	632
エラー 16305 .....	633
エラー 16306 .....	633
エラー 16307 ( 廃止予定 ) .....	633
エラー 16308 ( 廃止予定 ) .....	634
エラー 16309 .....	634
エラー 16310 .....	635
エラー 16311 ( 廃止予定 ) .....	635
エラー 16312 ( 廃止予定 ) .....	635
エラー 16313 ( 廃止予定 ) .....	636
エラー 16314 ( 廃止予定 ) .....	636
エラー 16315 .....	636
エラー 16316 ( 廃止予定 ) .....	637
エラー 16317 .....	637
エラー 16318 ( 廃止予定 ) .....	637
エラー 16319 .....	638
エラー 16320 .....	638
エラー 16321 .....	639
エラー 16322 .....	639
エラー 16323 .....	639
エラー 16324 .....	640



エラー 16325 .....	640
エラー 16326 .....	641
エラー 16327 .....	641
エラー 16328 .....	641
エラー 16329 .....	642
エラー 16330 .....	642
エラー 16331 .....	643
エラー 16332 .....	643
エラー 16333 .....	643
エラー 16334 .....	644
エラー 16335 .....	644
エラー 16336 .....	644
エラー 16337 .....	645
エラー 16338 .....	645
エラー 16339 .....	646
エラー 16340 .....	646
エラー 16341 (廃止予定) .....	646
エラー 16342 (廃止予定) .....	647
エラー 16343 (廃止予定) .....	647
エラー 16344 (廃止予定) .....	647
エラー 16345 (廃止予定) .....	647
エラー 16346 (廃止予定) .....	648
エラー 16347 (廃止予定) .....	648
エラー 16348 (廃止予定) .....	648
エラー 16349 (廃止予定) .....	649
エラー 16350 (廃止予定) .....	649
エラー 16351 (廃止予定) .....	649
エラー 16352 (廃止予定) .....	650
エラー 16353 (廃止予定) .....	650
エラー 16354 (廃止予定) .....	650
エラー 16355 (廃止予定) .....	650
エラー 16356 (廃止予定) .....	651
エラー 16357 (廃止予定) .....	651
エラー 16358 (廃止予定) .....	651
エラー 16359 (廃止予定) .....	652
エラー 16360 .....	652
エラー 16361 .....	652
エラー 16362 .....	653
エラー 16363 .....	653
エラー 16364 .....	654
エラー 16365 .....	654
エラー 16366 (廃止予定) .....	654
エラー 16367 .....	655
エラー 16368 .....	655
エラー 16369 .....	655

エラー 16370 ( 廃止予定 ).....	656
エラー 16371 .....	656
エラー 16372 .....	656
エラー 16373 .....	657
エラー 16374 .....	657
エラー 16375 .....	658
エラー 16376 ( 廃止予定 ).....	658
エラー 16377 ( 廃止予定 ).....	658
エラー 16378 .....	659
エラー 16379 .....	659
エラー 16380 .....	659
エラー 16381 .....	660
エラー 16382 .....	661
エラー 16383 .....	661
エラー 16384 ( 廃止予定 ).....	661
エラー 16385 .....	662
エラー 16386 .....	662
エラー 16387 ( 廃止予定 ).....	662
エラー 16388 .....	663
エラー 16389 .....	663
エラー 16390 .....	663
エラー 16391 .....	664
エラー 16392 .....	664
エラー 16393 .....	665
エラー 16394 .....	665
エラー 16395 .....	666
エラー 16396 .....	666
エラー 16397 .....	666
エラー 16398 .....	667

用語解説 .....	669
------------	-----

索引 .....	679
----------	-----

# はじめに

このマニュアルでは、Open Server™ Server-Library (C バージョン用) のリファレンス情報について説明します。

## 対象読者

『Open Server Server-Library/C リファレンス・マニュアル』は、Open Server アプリケーションを作成するプログラマを対象としたリファレンス・マニュアルです。このマニュアルは、C プログラミング言語に精通したアプリケーション・プログラマを対象としています。

## このマニュアルの内容

このマニュアルには、以下の章があります。

- 「[第 1 章 Open Server の概要](#)」では、Open Server の概要について説明します。
- 「[第 2 章 トピック名](#)」では、「サーバ」から text 値や image 値を読み込むために「**Server-Library**」ルーチンを使用するなど、特定のプログラミング・タスクを実行する方法について説明します。また、Open Server の構造体、プログラミング技法、エラー処理に関する情報も提供します。
- 「[第 3 章 ルーチン](#)」では、個々のルーチンが使用するパラメータや戻り値など、各 Server-Library ルーチンに関する固有の情報について説明します。
- 「[第 4 章 システム・レジスタード・プロシージャ](#)」では、Server-Library が自動的に提供するレジスタード・プロシージャについて説明します。また、パラメータ、結果、メッセージに関する情報も提供します。
- 「[付録 A Open Server エラー・メッセージ](#)」では、Open Server のエラー・メッセージの一覧を示し、それぞれについて説明します。

## 関連マニュアル

詳細については、これらのマニュアルを参照できます。

- 『Open Server および SDK 新機能 (Windows、Linux、UNIX 対応)』では、Open Server と Software Developer's Kit の新機能について説明しています。このマニュアルは、新機能の提供に伴って改訂されます。

- 
- 使用しているプラットフォームの『Open Server リリース・ノート』には、Open Server に関する重要な最新情報が記載されています。
  - 使用しているプラットフォームの『Software Developer’s Kit リリース・ノート』には、Open Client™ および SDK に関する重要な最新情報が記載されています。
  - 『jConnect™ for JDBC™ リリース・ノート』には、jConnect に関する重要な最新情報が記載されています。
  - 使用しているプラットフォームの『Open Client/Server 設定ガイド』では、システムを設定して Open Client/Server 製品を実行する方法について説明しています。
  - 『Open Client Client-Library/C プログラマーズ・ガイド』では、Client-Library アプリケーションの設計方法および実装方法について説明しています。
  - 『Open Client Client-Library/C リファレンス・マニュアル』では、Open Client Client-Library™ のリファレンス情報について説明しています。
  - 『Open Client および Open Server Common Libraries リファレンス・マニュアル』では、CS-Library のリファレンス情報について説明しています。CS-Library は、Client-Library と Server-Library の両方のアプリケーションで役に立つユーティリティ・ルーチンの集まりです。
  - 『Open Server DB-Library/C リファレンス・マニュアル』では、C バージョンの Open Client DB-Library™ のリファレンス情報について説明しています。
  - お使いのプラットフォーム用『Open Client/Server プログラマーズ・ガイド補足』では、Open Client/Server を使用するプログラマのために、プラットフォーム固有の情報について説明しています。このマニュアルには、次の情報が含まれています。
    - アプリケーションのコンパイルおよびリンク
    - Open Client/Server に含まれているサンプル・プログラム
    - プラットフォーム固有の動作をするルーチン

- 『Sybase® SDK DB-Library Kerberos 認証オプションのインストールおよびリリース・ノート』では、DB-Library で使用する MIT Kerberos セキュリティ・メカニズムをインストールして有効にする方法について説明しています。DB-Library でサポートされる Kerberos セキュリティ・メカニズムの機能は、ネットワーク認証サービスと相互認証サービスのみです。
- 『Open Client Client-Library 移行ガイド』には、Open Client™ DB-Library™ アプリケーションを Open Client Client-Library に移行する方法に関する情報が記載されています。
- 『Open Client/Server 開発者用国際化ガイド』では、国際化されたアプリケーションとローカライズされたアプリケーションを作成する方法について説明しています。
- 『Open Client Embedded SQL™/C プログラマーズ・ガイド』では、C アプリケーションで Embedded SQL および Embedded SQL プリコンパイラを使用する方法について説明しています。
- 『Open Client Embedded SQL™/COBOL プログラマーズ・ガイド』では、COBOL アプリケーションで Embedded SQL および Embedded SQL プリコンパイラを使用する方法について説明しています。
- 『jConnect for JDBC プログラマーズ・リファレンス』では、jConnect for JDBC 製品について説明し、リレーショナル・データベース管理システムに保管されているデータにアクセスする方法について説明しています。
- 『Adaptive Server® Enterprise ADO.NET Data Provider ユーザーズ・ガイド』では、C#、Visual Basic .NET、マネージ拡張を備えた C++、J# など、.NET でサポートされる任意の言語を使用して Adaptive Server 内のデータにアクセスする方法について説明しています。
- Sybase 製 *Adaptive Server Enterprise ODBC* ドライバの『ユーザーズ・ガイド』(Microsoft Windows および UNIX 版)では、Microsoft Windows および UNIX プラットフォームの Adaptive Server から、Open Database Connectivity (ODBC) ドライバを使用してデータにアクセスする方法について説明します。
- Sybase 製 *Adaptive Server Enterprise OLE DB* プロバイダの『ユーザーズ・ガイド』(Microsoft Windows 版)では、Microsoft Windows プラットフォームの Adaptive Server から、Adaptive Server OLE DB プロバイダを使用してデータにアクセスする方法について説明します。

- 『Perl 用 Adaptive Server Enterprise データベース・ドライバ・プログラマーズ・ガイド』では、Perl 開発者が Perl スクリプトを使用して Adaptive Server のデータベースに接続し、情報をクエリまたは変更する方法について説明しています。
- 『PHP 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』では、PHP 開発者が Adaptive Server データベースに対してクエリを実行する方法について説明しています。
- 『Python 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』では、Adaptive Server データベースに対してクエリを実行するときに使用できる Sybase 固有の Python インタフェースについて説明しています。

## その他の情報

Sybase Product Documentation Web サイトを使用して製品について詳しく知ることができます。

- Sybase Product Documentation Web サイトには、標準の Web ブラウザを使用してアクセスできます。また、製品ドキュメントのほか、EBFs/Maintenance、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

Sybase Product Documentation Web サイトは、Product Documentation (<http://www.sybase.com/support/manuals/>) にあります。

## Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

### ❖ 製品認定の最新情報にアクセスする

- 1 Web ブラウザで Technical Documents (<http://www.sybase.com/support/techdocs/>) を指定します。
- 2 [Partner Certification Report] をクリックします。
- 3 [Partner Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。
- 4 [Partner Certification Report] のタイトルをクリックして、レポートを表示します。

### ❖ コンポーネント認定の最新情報にアクセスする

- 1 Web ブラウザで Availability and Certification Reports (<http://certification.sybase.com/>) を指定します。
- 2 [Search By Base Product] で製品ファミリーとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。

3 [Search] をクリックして、入手状況と認定レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用カスタマイズできます。

- 1 Web ブラウザで Technical Documents (<http://www.sybase.com/support/techdocs/>) を指定します。
- 2 [MySybase] をクリックし、MySybase プロファイルを作成します。

## Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

- 1 Web ブラウザで the Sybase Support Page (<http://www.sybase.com/support>) を指定します。
- 2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。
- 3 製品を選択します。
- 4 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。

- 5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

## 表記規則

表 1：構文の表記規則

凡例	定義
コマンド	コマンド名、コマンドのオプション名、ユーティリティ名、ユーティリティのフラグ、キーワードは sans serif で示す。
変数	変数 (ユーザが入力する値を表す語) は斜体で表記する。

凡例	定義
{ }	中カッコは、その中から必ず1つ以上のオプションを選択しなければならないことを意味する。コマンドには中カッコは入力しない。
[ ]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには中カッコは入力しない。
( )	このカッコはコマンドの一部として入力する。
	中カッコまたは角カッコの中の縦線で区切られたオプションのうち1つだけを選択できることを意味する。
,	中カッコまたは角カッコの中のカンマで区切られたオプションをいくつでも選択できることを意味する。複数のオプションを選択する場合には、オプションをカンマで区切る。

## アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Open Client および Open Server のマニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

**注意** アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれませんが。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (<http://www.sybase.com/accessibility>) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

## 不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。



# Open Server の概要

トピック	ページ
クライアント／サーバ・アーキテクチャの概要	1
クライアントのタイプ	2
サーバのタイプ	2
Open Server の構成	4
Open Server	6
Open Server の使用	8
基本的な Open Server プログラム	10
Open Server イベント	14
レジスタード・プロシージャ	16
クライアントへの結果の送信	16
エラー処理	18
マルチスレッド・プログラミング	18

## クライアント／サーバ・アーキテクチャの概要

クライアント／サーバ・アーキテクチャは、コンピューティング作業をクライアントとサーバ間で分担します。

クライアントはサーバに対して要求を行い、サーバから返された要求の結果を処理します。たとえば、あるクライアント・アプリケーションはデータベース・サーバの温度データを要求し、部屋の温度を下げるよう環境制御サーバに要求するアプリケーションもあります。

サーバは、データやその他の情報をクライアントへ返したり、何らかのアクションをとったりして、要求に応答します。たとえば、データベース・サーバは、表形式のデータとそのデータについての情報をクライアントに返し、電子メール・サーバは、受け取ったメールを最終的な宛先に転送します。

クライアント／サーバ・アーキテクチャは、次の点が従来のプログラム・アーキテクチャよりも優れています。

- 共通するサービスが1か所で、つまり1つのサーバで処理されるため、アプリケーションの大きさや複雑さがかなり軽減されます。これによって、クライアント・アプリケーションが単純化され、コードの重複が減り、アプリケーションの保守が容易になります。
- クライアント/サーバ・アーキテクチャは、さまざまなアプリケーションの間の通信を容易にします。異なった通信プロトコルを使用するクライアント・アプリケーション同士は、そのままの状態では直接通信できませんが、両方のプロトコルを「認識する」サーバを介せば通信できるようになります。このようなサーバを「ゲートウェイ」と呼びます。
- クライアント/サーバ・アーキテクチャを使用すると、アプリケーションを、独立した個々のコンポーネントの集合体として開発できます。これらのコンポーネントは、アプリケーションの他の部分に影響を与えずに、変更したり置き換えたりできます。

## クライアントのタイプ

クライアントとは、サーバへ要求を行うアプリケーションのことです。Sybase のクライアントには、次のものが含まれています。

- Sybase SQL Toolset™ 製品
- isql および bcp など、Adaptive Server Enterprise が提供するスタンドアロン・ユーティリティ
- Open Client ライブラリを使用して作成されたアプリケーション
- Embedded SQL™ を使用して記述されたアプリケーション
- PowerBuilder® アプリケーション

## サーバのタイプ

Sybase の製品群には、次に示すように、サーバとサーバを構築するためのツールが含まれています。

- Adaptive Server Enterprise はデータベース・サーバです。Adaptive Server Enterprise は、1 つまたは複数のデータベースに格納された情報を管理します。

- Open Server はカスタム・サーバの作成に必要なツールやインタフェースを提供します。Open Server で構築したカスタム・サーバを「Open Server アプリケーション」と呼びます。

Open Server アプリケーションは、あらゆる種類のサーバとして機能します。たとえば、Open Server アプリケーションは、特殊な計算を実行したり、リアルタイムでデータにアクセスしたり、電子メールのようなサービスに対するインタフェースとして機能することができます。Open Server アプリケーションは、Open Server Server-Library から提供されるルーチンを組み合わせて個々に開発します。

Adaptive Server Enterprise と Open Server アプリケーションには、次のような類似点があります。

- Adaptive Server Enterprise と Open Server はどちらもサーバであり、クライアントの要求に応答する。
- クライアントは Open Client ライブラリを介して、Adaptive Server Enterprise と Open Server の両方のアプリケーションと通信する。

しかし、次のような相違点もあります。

- アプリケーション・プログラマは、Open Server から提供されるルーチンを使用してカスタム・コードを開発し、Open Server アプリケーションを作成する必要がある。Adaptive Server Enterprise は完成されており、カスタム・コードは必要ありません。
- Open Server アプリケーションは、あらゆる種類のサーバとして使用できるため、さまざまなプログラム言語を使用できるように作成できる。Adaptive Server Enterprise はデータベース・サーバであり、Transact-SQL のみを認識します。
- Open Server アプリケーションは、Sybase の Tabular Data Stream™ (TDS) プロトコルに基づいていない「外部」のアプリケーションやサーバと通信できる。また、Sybase のアプリケーションやサーバとも通信できる。Adaptive Server Enterprise が直接通信できるのは、Sybase のアプリケーションとサーバだけである。外部アプリケーションや外部サーバと通信する場合、Adaptive Server Enterprise は Open Server のゲートウェイ・アプリケーションを仲介として使用しなくてはならない。

## Open Server の構成

クライアント/サーバ・アーキテクチャにおける Open Server アプリケーションの位置付けは、Open Server がどのように機能しているかによって異なります。Open Server アプリケーションの機能は、次の3つに分類されます。

- スタンドアロン
- 補助
- ゲートウェイ

### スタンドアロン Open Server アプリケーション

クライアントは、スタンドアロン Open Server アプリケーションに直接接続します。

クライアントは、次の方法を使用して要求をサーバに送信します。

- リモート・プロシージャ・コール (RPC) – RPC の呼び出しによって、Open Server アプリケーションで「**レジスタード・プロシージャ**」を実行できます。レジスタード・プロシージャは、Open Server アプリケーションに格納される Open Server のコードの一部として定義されます。これらは、ユーザ定義またはシステム定義のプロシージャです。
- 「**カーソル**」コマンド
- さまざまなクライアント「**コマンド**」

Open Server アプリケーションのプログラマは、クライアント・コマンドを処理するコードを記述します。

スタンドアロン Open Server アプリケーションは、クライアント要求に応答するために外部要求を行いません。

### 補助 Open Server アプリケーション

補助 Open Server アプリケーションは、リモート・プロシージャ・コール (RPC) を処理することによって Adaptive Server Enterprise をサポートします。

クライアントは、Transact-SQL を使用し、Adaptive Server Enterprise に直接接続します。Open Server アプリケーション上でレジスタード・プロシージャを実行する場合、クライアントは、Transact-SQL 文のプロシージャ名のプレフィクスとして、使用する Open Server アプリケーション名を指定します。Adaptive Server Enterprise は、この名前を使用して RPC を開始します。たとえば、次のクライアント文では、“OpnSrv211” という名前の Open Server アプリケーション上で “print\_calls” プロシージャを実行します。

```
exec OpnSrv211...print_calls
```

RPC は、Adaptive Server Enterprise から Open Server アプリケーションに直接送信できる唯一のクライアント・コマンドです。Adaptive Server Enterprise 内でストアド・プロシージャ、トリガ、またはスレッシュホールド管理を使用して RPC 呼び出しを開始できます。RPC を使用すると、次の機能にアクセスできます。

- 電子メールの送信と印刷などのオペレーティング・システム機能
- Open Server アプリケーションのコードに定義したすべての機能

Open Server アプリケーションは、Adaptive Server Enterprise に情報を返したり、Adaptive Server Enterprise を経由してクライアントに情報を戻したりすることができます。

サーバ間の RPC を使用することによって、Open Server アプリケーションは、特殊な計算を実行したり、リアルタイムでデータにアクセスしたり、電子メールなどのサービスへのアクセスを Adaptive Server Enterprise に許可したりすることができます。

## ゲートウェイ Open Server アプリケーション

ゲートウェイ・サーバを使用すると、クライアントは、クライアント接続を直接受け入れるかどうかわからないサーバにもアクセスできます。ゲートウェイは、Adaptive Server Enterprise などの DBMS に接続する必要はありません。ファイル・システムや、サーバとして動作可能なアプリケーション・プログラムであれば、ゲートウェイを接続できます。

Adaptive Server Enterprise や他の Open Server アプリケーションにアクセスする Open Server アプリケーションは、Client-Library ルーチンと Server-Library ルーチンの両方を含んでいます。両方のルーチンを含むことで、クライアントとサーバのどちらの役割も果たすことができます。サーバの役割としては、クライアントとのインタフェースとして Open Server を使用します。クライアントの役割としては、Client-Library ルーチンを使用して、Adaptive Server Enterprise や他の Open Server に要求を送信したり、結果を受け取ったりします。詳細については、「[ゲートウェイ・アプリケーション](#)」(109 ページ) を参照してください。

上記のゲートウェイは、クライアントを Adaptive Server Enterprise に接続します。図内の点線は、この特定のゲートウェイが Sybase クライアントと Sybase サーバ間で要求と結果を受け渡すオーバーヘッドの小さい方法である “TDS バススルー モード” を使用することを示しています。詳細については、「[バススルー・モード](#)」(146 ページ) を参照してください。

## Open Server

Open Server では、カスタム・サーバ・アプリケーションの作成に必要なツールとインタフェースを提供します。

Open Server は、大きく分けて、関数のライブラリとしてのプログラミング・インタフェースとネットワーク・サービスで構成されます。

## Open Server ライブラリ

Open Server プログラミング・インタフェースを構成しているライブラリは、次のとおりです。

- **Server-Library** : サーバ・アプリケーションの作成に使用するルーチンの集まりです。Server-Library には、次のルーチンが含まれます。
  - クライアントから送信されたコマンドを受信するルーチン
  - 結果をクライアントに返すルーチン
  - アプリケーション属性を設定するルーチン
  - エラー条件を処理するルーチン

- クライアントとの対話をスケジュールするルーチン
- クライアント接続についてのさまざまな情報を提供するルーチン
- **CS-Library** : クライアント・アプリケーションとサーバ・アプリケーションの両方に役立つユーティリティ・ルーチンの集まりです。**Server-Library** ルーチンは **CS-Library** に割り当てられている構造体を使用しているため、すべての **Server-Library** プログラムは **CS-Library** への呼び出しを最低 1 つは含んでいなければなりません。

(Open Client と Open Server はともに **CS-Library** を使用しますが、この **CS-Library** には、クライアント・アプリケーションとサーバ・アプリケーションの両方のユーティリティ・ルーチンが入っています。)

スタンドアロン Open Server アプリケーション、および補助 Open Server アプリケーションには、**Server-Library** と **CS-Library** への呼び出しが含まれています。ゲートウェイ Open Server アプリケーションには、**Server-Library**、**CS-Library**、**Client-Library** への呼び出しが含まれています。

Open Server には、**Server-Library** ルーチンが使用する構造体、型、値を定義する次のヘッダ・ファイルが含まれています。含まれているヘッダ・ファイルは次のとおりです。

- *ospublic.h*
- *oserror.h*
- *oscompat.h*

## ネットワーク・サービス

通常、Open Server のネットワーク・サービスは、Open Server 開発者や Open Server アプリケーションのエンド・ユーザにとって透過的で意識されません。ただし、PC プラットフォームではネットワーク・サービスは外部に存在し、意識されます。

ネットワーク・サービスには **Net-Library** があり、TCP/IP などの、特定のネットワーク・プロトコルをサポートします。

## Open Server の使用

Server-Library および CS-Library への呼び出しを使用して、構造体の設定、クライアントや他のサーバからの接続要求の受信、クライアントからの要求の処理、メモリのクリーンアップを行い、Open Server アプリケーションを作成します。ゲートウェイ・アプリケーションにも、Client-Library ルーチンへの呼び出しが含まれています。

Open Server アプリケーション・プログラムは、C 言語プログラムと同じようにコンパイルされます。ほとんどの UNIX プラットフォームでは、プログラムのコンパイルやリンクを実行するときには次のライブラリが必要です(ファイル名や拡張子はプラットフォームによって異なります)。

- *libsybsrv.a*
- *libsybcs.a*
- *libsybcomn.a*
- *libsybtcl.a*
- *libsybintl.a*
- *libsybblk.a* – バルク・コピー・ルーチンを使用している場合
- *libsybct.a* – ゲートウェイを使用している場合

ライブラリ・ファイルは、`$SYBASE/$SYBASE_OCS/lib` ディレクトリにあります。

## CS\_CONTEXT 構造体

Open Server アプリケーションには、特定のアプリケーション「コンテキスト」やオペレーション環境を定義する CS\_CONTEXT 構造体が必要です。CS\_CONTEXT には、サーバワイドな制御情報だけでなく、ローカライゼーション情報も含まれています。どのような Open Server アプリケーション・プログラムでも、最初に `cs_ctx_alloc` を呼び出して CS\_CONTEXT 構造体を割り付けます。

アプリケーション・プログラムは、CS\_CONTEXT 構造体の内容进行操作することによってアプリケーションの動作や属性を決定します。[「プロパティ」\(157 ページ\)](#) を参照してください。



## 簡単なプログラムの手順

ほとんどのプラットフォームでは、次の手順に従って、簡単な Open Server アプリケーション・プログラムを作成できます。

手順	機能	ルーチン
1	構造体の割り付け、および「プロパティ」と呼ばれるグローバル属性の設定を行い、Open Server オペレーティング環境を設定する。	cs_ctx_alloc srv_version srv_props
2	エラー処理を定義する。アプリケーションにエラー処理ルーチンをインストールし、エラーが検出されたときに Open Server がこのルーチンを呼び出すようにする。アプリケーションは、アドホック・ベースで <code>srv_sendinfo</code> ルーチンを呼び出してクライアントにエラー・メッセージを送信することもできる。また、 <code>srv_log</code> を呼び出してログ・ファイルに書き込むこともできる。詳細については、「エラー」(97 ページ)を参照してください。	srv_props(SRV_S_ERRHANDLE)
3	サーバを初期化する。	srv_init
4	クライアント・コマンドが Open Server イベントをトリガするときに Open Server が呼び出すイベント処理ルーチンをインストールする。Open Server アプリケーションの作業のほとんどが、イベント処理ルーチンの中で行われる。「Open Server イベント」(14 ページ)を参照してください。	srv_handle
5	サーバを起動させる。この状態では、サーバはクライアント要求を受信するだけである。	srv_run
6	クリーンアップし、終了する。	cs_ctx_drop

次の項のサンプル・プログラムには、手順 4 を除くすべての手順を示します。この例では、ユーザ定義のイベント・ハンドラをインストールするのではなく、デフォルトのハンドラを実行します。

## 基本的な Open Server プログラム

以下に、基本的な Open Server アプリケーション・プログラムのコードを示します。

```
/*
** This program demonstrates the minimum steps necessary
** to initialize and start up an Open Server application.
** No user-defined event handlers are installed, therefore
** the default handlers will execute instead.
*/

/*
** Include the required Open Server header files.
**
**  opublic.h: Public Open Server structures, typedefs,
**  defines, and function prototypes.
**
**  oerror.h: Open Server error number #defines. This header
**  file is only required if the Open Server application wants
**  to detect specific errors inside the Open Server error
**  handler.
*/

#include      <opublic.h>
#include      <oerror.h>

/*
** Include the operating system specific header files required
** by this Open Server application.
*/
#include <stdio.h>

/*
** Local defines.
**
**  OS_ARGCOUNT Expected number of command line arguments
*/
#define      OS_ARGCOUNT      2

/*
** This Open Server application expects the following
** command line arguments:
**
**  servername: The name of the Open Server application.
**
** This name must exist in the interfaces file defined by
```

```
**      the SYBASE environment variable.
**
** Returns:
**      0      Open Server exited successfully.
**      1      An error was detected during initialization.
*/

int      main(argc, argv)
int      argc;
char     *argv[];
{
    CS_CONTEXT    *cp;          /* Context structure */
    CS_CHAR       *servername;  /* Open Server name */
    CS_CHAR       logfile[512]; /* Log file name */
    CS_BOOL       ok;          /* Error control flag */
    SRV_SERVER    *ssp;        /* Server control structure*/

    /* Initialization.*/
    ok = CS_TRUE;

    /*
    ** Read the command line options. There must be one
    ** argument specifying the server name.
    */

    if(argc != OS_ARGCOUNT)
    {
        (CS_VOID)fprintf(stderr, "Invalid number of
        arguments(%d) %n", argc);

        (CS_VOID)fprintf(stderr, "Usage:<program>
        <server name>%n");
        exit(1);
    }

    /*
    ** Initialize 'servername' to the command line argument
    ** provided.
    */

    servername = (CS_CHAR *)argv[1];

    /*
    ** Allocate a CS-Library context structure to define the
    ** default localization information. Open Server
```

```
** also stores global state information in this structure
** during initialization.
*/
if(cs_ctx_alloc(CS_VERSION_155, &cp) != CS_SUCCEED)
{
    (CS_VOID)fprintf(stderr, "%s:cs_ctx_alloc failed",
        servername);
    exit(1);
}

/*
** Default Open Server localization information can be
** changed here before calling srv_version, using cs_config
** and cs_locale.
*/

/*
** Set the Open Server version and context information
*/
if(srv_version(cp, CS_VERSION_155) != CS_SUCCEED)
{
    /*
    ** Release the context structure already allocated.
    */
    (CS_VOID)cs_ctx_drop(cp);

    (CS_VOID)fprintf(stderr, "%s:srv_version failed",
        servername);
    exit(1);
}

/*
** There is no error handler installed in this sample
** Open Server application.Any errors detected by Open
** Server are written to the Open Server log file
** configured below.A real Open Server application would
** install an error handler after calling srv_version, using
** srv_props(SRV_S_ERRHANDLE).Then, any subsequent errors
** will be detected by the Open Server application code.
*/

/*
** Default Open Server global properties can be changed here
** before calling srv_init. We choose just to change the
** default log file name to use the name of this Open
** Server application.
```

```
*/

/*
** Build a new Open Server log file name using 'servername'
*/
(CS_VOID)sprintf(logfile, "%s.log", servername);

/*
** Set the new log file name using the global SRV_S_LOGFILE
** property.
*/
if(srv_props(cp, CS_SET, SRV_S_LOGFILE, logfile,
CS_NULLTERM,(CS_INT *)NULL) != CS_SUCCEEDED)
{
    /*
    ** Release the context structure already allocated.
    */
    (CS_VOID)cs_ctx_drop(cp);

    (CS_VOID)fprintf(stderr, "%s:srv_props(SRV_S_LOGFILE)
failed¥n", servername);
    exit(1);
}

/*
** Initialize Open Server.This causes Open Server to
** allocate internal control structures based on the global
** properties set above.Open Server also looks up
** the application name in the interfaces file.
*/
if((ssp = srv_init((SRV_CONFIG *)NULL, servername,
CS_NULLTERM)) == (SRV_SERVER *)NULL)
{
    /*
    ** Release the context structure already allocated
    */
    (CS_VOID)cs_ctx_drop(cp);

    (CS_VOID)fprintf(stderr, "%s:srv_init failed¥n",
servername);
    exit(1);
}

/*
** Start the Open Server application running.We don't
** install any event handlers in this simple example.This
```

```

** causes Open Server to use the default event handlers.
**
** The call to srv_run does not return until a fatal error is
** detected by this Open Server application, or a SRV_STOP
** event is queued. Since we haven't installed any event
** handlers, the only way to stop this Open Server
** application is to kill the operating system process in
** which it is running.
*/
if(srv_run((SRV_SERVER *)NULL) == CS_FAIL)
{
    (CS_VOID)fprintf(stderr, "%s:srv_run failed¥n",
        servername);
    ok = CS_FALSE;
}

/*
** Release all allocated control structures and exit.
*/
(CS_VOID)srv_free(ssp);
(CS_VOID)cs_ctx_drop(cp);
exit(ok ? 0 : 1);
}

```

## Open Server イベント

クライアントが Open Server アプリケーションに要求を送信すると、サーバ内の「イベント」がトリガされます。これによって、クライアントのサーバ・プロセス、つまり「スレッド」が、イベントを処理するルーチンを実行します。このルーチンを「イベント・ハンドラ」と呼びます。

Server-Library を使用して、さまざまな標準イベントが内部的に定義されます。次の表には、代表的な標準イベントを示します。

クライアント要求	イベント・タイプ	Open Server イベント
ct_command(CS_LANG_CMD) ct_send	言語	SRV_LANGUAGE
ct_command(CS_RPC_CMD) ct_send	RPC	SRV_RPC

クライアント要求	イベント・タイプ	Open Server イベント
ct_cancel	アテンション	SRV_ATTENTION
ct_connect	接続	SRV_CONNECT
ct_close	切断	SRV_DISCONNECT
ct_exit		
クライアントからは開始されない	開始	SRV_START
クライアントからは開始されない	停止	SRV_STOP

「イベント」(100 ページ)を参照してください。

## デフォルトのイベント・ハンドラ

標準イベントにはデフォルトのイベント・ハンドラがありますが、通常は、ユーザ自身がコーディングしたイベント・ハンドラに置き換えます。ほとんどのデフォルトのイベント・ハンドラは、単に要求をそのまま返すだけです。たとえば、デフォルトの言語イベント・ハンドラは、次のメッセージを返します。

```
No language handler installed.
```

あるイベント・ハンドラをインストールすると、デフォルトのイベント・ハンドラは自動的に上書きされます。

## クライアントでは開始されないイベント

次に示すイベントのように、クライアント・プログラムではトリガできないイベントもあります。

- ユーザ定義イベント
- Open Server のコードに `srv_event` を記述して呼び出すとトリガされる `SRV_STOP`
- 起動プロセスの一部として発生する `SRV_START`

## レジスタード・プロシージャ

レジスタード・プロシージャとは、名前で識別できる Open Server/C コードの 1 つです。アプリケーションはプロシージャを登録するときに、プロシージャ名をルーチンにマップします。これにより、Open Server は、受信 RPC データ・ストリーム内でそのプロシージャ名を検出すると、SRV\_RPC イベントを発生させずに特定のルーチンをただちに呼び出すことができます。

Open Server アプリケーションは RPC を受信すると、レジスタード・プロシージャのリスト内でそのプロシージャ名を検索します。プロシージャ名が登録されている場合、ランタイム・システムは、そのレジスタード・プロシージャと対応するルーチンを実行します。プロシージャ名がレジスタード・プロシージャのリストに見つからない場合には、Open Server は SRV\_RPC イベント・ハンドラを呼び出します。

システム・レジスタード・プロシージャは、すべての Open Server アプリケーションに組み込まれているプロシージャです。個々のシステム・レジスタード・プロシージャの詳細については、「[第 4 章 システム・レジスタード・プロシージャ](#)」を参照してください。

レジスタード・プロシージャの詳細については、「[レジスタード・プロシージャ](#)」(185 ページ)を参照してください。

## クライアントへの結果の送信

この項では、クライアントに対して送受信できる結果データの種類と順序について説明します。

### 結果データの種類

Open Server アプリケーションからクライアントに送信される結果は、次のとおりです。

- メッセージ
- データ・ロー
- 結果パラメータ
- ステータス値



1 つのクライアント要求には、2 つ以上の結果セットを指定できます。最初の結果セットを送信した後で、その要求に対してさらに結果セットがある場合には、`SRV_DONE_MORE` ステータスを使用して `srv_senddone` を呼び出します。それ以上結果がない場合は、`SRV_DONE_FINAL` ステータスを使用して `srv_senddone` を呼び出してください。`SRV_DONE_FINAL` ステータスを使用して `srv_senddone` を呼び出した場合は、クライアント要求への応答が最小限になります。クライアントは、`srv_senddone(SRV_DONE_FINAL)` を受信した後で処理を実行します。

## メッセージ

アプリケーションは、`srv_sendinfo` でクライアントにエラー・メッセージを送信できます。Client-Library プログラムは、メッセージ・ハンドラ・ルーチンでメッセージを処理します。このルーチンは、通常、メッセージ情報をユーザの端末に表示します。メッセージの種類が「エラー・メッセージ」の場合、クライアント・プログラムは、エラーからリカバリを試みることも終了することもできます。

## データ・ロー

Adaptive Server Enterprise が SQL クエリの結果を返すのと同様に、Open Server はデータ・ローをクライアントに返すことができます。ローは、1 つ以上のデータ・カラムで構成されています。詳細については、「[パラメータとロー・データの処理](#)」(152 ページ)を参照してください。

## パラメータ

パラメータは、クライアントと Open Server アプリケーションとの間で、クライアント・コマンドを使用して通信されるデータです。

## ステータス値

アプリケーションは `srv_sendstatus` を呼び出して、オプションのステータス値をクライアント・アプリケーションに返すことができます。ステータスは、アプリケーション固有の意味を持つ `CS_INT` 値です。`CS_INT` は、Open Server のデータ型です。「[データ型](#)」(229 ページ)を参照してください。1 つの結果セットには、ステータスは 1 つしかありません。

## 結果の順序

クライアントに結果を返す順序は重要です。

- データ・ローのセットには、他の種類の結果を割り込ませないでください。データ・ローは、データ・ロー全体がクライアントに送信されるまで、続けて送信する必要があります。たとえば、いくつかのデータ・ローを送信できない場合には、メッセージを送信してから、さらにローを送信します。
- 存在するすべてのデータ・ローについて送信を完了すると、メッセージやステータス情報を順序に関係なくクライアントに送信できます。
- 結果セットの最後に、結果が終わったことを伝える `srv_senddone` ルーチンを呼び出します。

## エラー処理

Open Server アプリケーションで最初に行う必要があるアクションの1つが、`srv_props` でエラー・ハンドラをインストールすることです。エラー・ハンドラがインストールされていない場合には、Open Server はエラー・メッセージをログ・ファイルに書き込みます。詳細については、「[エラー](#)」(97 ページ) を参照してください。

## マルチスレッド・プログラミング

Open Server は、マルチスレッド・アーキテクチャを備えています。このアーキテクチャによって、アプリケーション開発者はマルチスレッド・サーバを作成することができます。マルチスレッド・サーバとは、それぞれが独自のタスクを達成するためにルーチンを実行するスレッドの集合です。たとえば、各クライアントは、その接続を管理するスレッドを使用し、要求を満たすイベント・ハンドラとプロシージャを実行します。Open Server ランタイム・システムは、メッセージの転送、ネットワーク通信処理、サーバ内のタスクのスケジューリングなどのサーバ・アクティビティを管理するためのスレッドをいくつか持っています。他の非クライアント・アクティビティのためにスレッドを「発生」させることができます。

詳細については、「[マルチスレッド・プログラミング](#)」(123 ページ) を参照してください。

# トピック名

この章では、次の項目について説明します。

- パラメータとロー・データの処理、`text` および `image` のサポートなどの `Open Server` のプログラミングについてのトピック
- カーソル要求への応答やエラー処理など、具体的なプログラミング・タスクの実行に `Open Server` のルーチンを使用する方法
- `Open Server` のプロパティ、データ型、構造体

この章の内容は、次のとおりです。

トピック	ページ
<a href="#">アテンション・イベント</a>	20
<a href="#">ブラウザ・モード</a>	23
<a href="#">機能</a>	25
<a href="#">クライアント・コマンド・エラー</a>	38
<a href="#">接続マイグレーション</a>	41
<a href="#">CS_BROWSEDESC 構造体</a>	55
<a href="#">CS_DATAFMT 構造体</a>	57
<a href="#">CS_IODESC 構造体</a>	61
<a href="#">CS-Library</a>	62
<a href="#">CS_SERVERMSG 構造体</a>	64
<a href="#">カーソル</a>	66
<a href="#">スクロール可能カーソル</a>	83
<a href="#">データ・ストリーム・メッセージ</a>	86
<a href="#">ディレクトリ・サービス</a>	88
<a href="#">動的 SQL</a>	90
<a href="#">動的なリスナ</a>	95
<a href="#">エラー</a>	97
<a href="#">イベント</a>	100
<a href="#">ゲートウェイ・アプリケーション</a>	109
<a href="#">国際化サポート</a>	111
<a href="#">言語呼び出し</a>	120
<a href="#">ログイン・リダイレクトと拡張 HA フェールオーバーのサポート</a>	121

トピック	ページ
<a href="#">メッセージ</a>	122
<a href="#">マルチスレッド・プログラミング</a>	123
<a href="#">ネゴシエートされた動作</a>	135
<a href="#">オプション</a>	138
<a href="#">パラメータ</a>	143
<a href="#">部分更新</a>	144
<a href="#">パススルー・モード</a>	146
<a href="#">パラメータとロー・データの処理</a>	152
<a href="#">プロパティ</a>	157
<a href="#">レジスタード・プロシージャ</a>	185
<a href="#">リモート・プロシージャ・コール</a>	192
<a href="#">ラージ・オブジェクト・パラメータの取得</a>	194
<a href="#">セキュリティ・サービス</a>	196
<a href="#">text と image</a>	225
<a href="#">データ型</a>	229

## アテンション・イベント

クライアント・アプリケーションが `dbcancel` または `ct_cancel` コマンドを使用して要求をキャンセルすると、Open Server `SRV_ATTENTION` イベントがトリガされます。次に、Open Server によって、Open Server アプリケーションの `SRV_ATTENTION` イベント・ハンドラが呼び出されます。呼び出された `SRV_ATTENTION` イベント・ハンドラが返されると、Open Server は、アテンション・イベントが検出された時点で中断された位置から処理を再開します。

## 割り込みレベルのアクティビティ

`SRV_ATTENTION` イベント・ハンドラは、割り込みレベルで動作する唯一のイベント・ハンドラです。Open Server アプリケーションは、`SRV_ATTENTION` の中から次の Server-Library 呼び出しのみを発行できます。

- `wakeflags` 引数として `SRV_M_WAKE_INTR` を設定した `srv_wakeup`

- `wakeflags` 引数として `SRV_M_WAKE_INTR` を設定した `srv_ucwakeup`
- `cmd` 引数として `CS_GET` を設定した `srv_thread_props`
- `cmd` 引数として `CS_GET` を設定した `srv_props`
- `srv_event_deferred`

上記以外の `Server-Library` ルーチンの場合は、`SRV_ATTENTION` イベント・ハンドラや他の割り込みレベルのコードを使用して呼び出すことはできません。

## アテンション・イベントに関するコーディングの考慮事項

アテンション・イベントが問題となるのは、非割り込みレベルのハンドラのコードを実行しているときにアテンション・イベントが到着する場合です。クライアントが要求をキャンセルしたために、アプリケーションが、必要のなくなった作業を行う可能性があります。

非割り込みレベルで、時間のかかる I/O タスクや計算を中心とした作業を実行している場合は、アプリケーションがアテンション・イベントを定期的にチェックする必要があります。アプリケーション・コードは、`srv_thread_props` を使用してアテンション・イベントを定期的にチェックする必要があります。このとき、`cmd` に `CS_GET` を、`property` に `SRV_T_GOTATTENTION` を設定します。

アテンション・イベントを検出すると、`Open Server` アプリケーション・コードは引き続き結果を送信できますが、クライアントはこれを無視します。アプリケーションがアテンション・イベントに応答する最も簡単な方法は、クライアントに `SRV_DONE_FINAL` を送信して戻ることです。

ゲートウェイ・アプリケーション・コードの中の `Client-Library` 部分が実行されているときに、アテンション・イベントが到着する可能性があります。アプリケーションは、`SRV_ATTENTION` イベント・ハンドラで `type` 引数を `CS_CANCEL_ATTEN` に設定して `ct_cancel` を呼び出し、`Client-Library` ルーチンを非割り込みレベルのコードに戻します。アテンション・イベントが到着していない場合はこのコマンドは無効なので、ゲートウェイ・アプリケーションは、このコマンドを定期的に呼び出す必要があります。

クライアント I/O を行うすべてのゲートウェイ呼び出しは、`srv_thread_props` を使用してアテンション・イベントをチェックしてから、`ct_send` を呼び出します。このプロシージャによって、クライアントがキャンセルした「クエリ」がリモート・サーバに送信されることはありません。

## 切断の処理

Open Server アプリケーションがクライアントに結果を送信している途中で、クライアント接続が突然切断された場合、アプリケーションは、接続がクローズされたことを検出するまで引き続き結果を送信します。続いて、Open Server は `SRV_DISCONNECT` イベント・ハンドラを呼び出します。この場合は、すでに受信できない状態にあるクライアントに対して、アプリケーションが結果を送信し続けていることとなります。次のような場合、クライアント接続が突然切断されることがあります。

- サーバから送信されるすべての結果をクライアントが処理する前に `ct_close` を呼び出した場合
- クライアントが処理を突然中断した場合
- マシンがダウンした場合

この事態を避けるために、アプリケーションは、クライアント接続の切断に対して Open Server が最初にアプリケーションの `SRV_ATTENTION` イベント・ハンドラを呼び出した後、`SRV_DISCONNECT` イベント・ハンドラを呼び出すように要求できます。Open Server がこのような形で切断を処理するには、アプリケーションは `srv_props` を使用して `SRV_S_DISCONNECT` プロパティを `CS_TRUE` に設定する必要があります。 `SRV_DISCONNECT` イベント・ハンドラは通常どおりに呼び出されますが、`SRV_ATTENTION` ハンドラの後に呼び出されます。 `SRV_S_DISCONNECT` プロパティのデフォルトは、`CS_FALSE` です。

`SRV_ATTENTION` ハンドラは、I/O アクティビティを終了したり、切断時に実行中だったルーチンからの結果の応答を停止したりするための、適切な手順を開始します。このように、アプリケーションは、アテンションに応答する場合と同様に切断にも応答することができます。

アプリケーションは、`SRV_ATTENTION` イベント・ハンドラを使用して、アテンションと接続の切断のどちらがハンドラをトリガしたのかを判断することもできます。そのためには、`cmd` を `CS_GET` に、`property` を `SRV_S_ATTREASON` に設定して `srv_props` を呼び出します。

## 例

サンプル・プログラム *ctos.c* には、アテンション処理のコードが記述されています。

## ブラウズ・モード

---

**注意** ブラウズ・モードは、バージョン 11.1 以降の Open Client ライブラリとの互換性を実現するためのモードです。Sybase では、新しい Open Server Server-Library アプリケーションでブラウズ・モードを使用することをおすすめしません。ブラウズ・モードと同じ機能性を備えており、ブラウズ・モードよりポータブルで柔軟性のあるカーソルを使うことをおすすめします。さらに、ブラウズ・モードは Sybase 固有のものであり、異機種接続環境での使用には適していません。

---

ブラウズ・モードを使用すると、データベース・ローを検索し、その検索結果の値をロー単位で更新できます。ただし、その処理を実施するには、クライアント・アプリケーション内の処理としていくつかの手順が必要になります。その理由は、各ローをデータベースからクライアント・アプリケーション・プログラム用の変数に変換してからでないと、ローを検索したり更新したりできないからです。

ブラウズされているローはデータベースに常駐している実際のローではなく、プログラム変数に常駐しているコピーなので、プログラムは、変数の値になされた変更に基づいて元のデータベース・ローを更新しなければなりません。マルチユーザ環境の場合は、あるユーザがデータベースを変更しても、それ以前に別のユーザによって変更された内容が書き換えられることがないように、プログラムによる保証が必要です。このような書き換えが起こるのは、クライアント・アプリケーションはデータベース内の複数のローを一度に選択して更新するのに対して、アプリケーションのユーザはデータベース内のローを一度に1つずつ検索して更新するためです。ブラウズ可能なテーブル内の `timestamp` カラムが、このようなマルチユーザの更新を調整するのに必要な情報を提供します。

アドホックのブラウズ・モード・クエリの入力をユーザに許可するクライアント・アプリケーションでは、ユーザ・コマンドによってテーブルの内容が変更された場合は、基本データベース・テーブルを更新しなければなりません。そのため、これらのアプリケーションには、ブラウズ・モード・コマンドの基本構造体についての情報が必要な場合があります。

Open Server には、そのような情報を提供する 2 つのルーチン、`srv_tabname` と `srv_tabcolname` が用意されています。

- `srv_tabname` は、ブラウズ・モード・コマンドに含まれる各テーブルの名前と番号を返します。
- `srv_tabcolname` は、`CS_BROWSEDESC` 構造体を介して、結果カラムに関するさまざまな情報を返します。[「CS\\_BROWSEDESC 構造体」\(55 ページ\)](#) を参照してください。

Open Server アプリケーションがブラウズ・モード要求を受け取ると、標準のデータ・バインド・ルーチンに従ってブラウズ・モード情報を返すために、上記の 2 つのルーチン呼び出すことができます。具体的な手順は、次のとおりです。

- 1 結果ローのソースとなるテーブルごとに、`srv_tabname` を 1 回ずつ呼び出します。
- 2 結果ロー内のカラムごとに、`srv_descfmt` と `srv_tabcolname` をこの順序で 1 回ずつ呼び出します。

Open Server アプリケーションで `CS_BROWSEDESC` 構造体の `status` フィールドが `CS_RENAMED` に設定されている場合は、クライアント・アプリケーションのブラウズ・モードの `select` 文によってカラム名が変更されています。Open Server アプリケーションでは、`CS_BROWSEDESC` 構造体の `origname` フィールドと `origlen` フィールドにデータベース内のカラムのオリジナル名およびカラム名の長さを入力してから、`srv_tabcolname` を呼び出す必要があります。

- 3 カラム・データのバインドには `srv_bind` ルーチンを、転送には `srv_xferdata` ルーチンを使用します。

---

**注意** `srv_tabcolname` には `srv_tabname` によって返される情報 (ユニークなテーブル番号) が必要なため、先に `srv_tabname` を呼び出したあとで `srv_tabcolname` を呼び出します。

---

『Open Client Library/C リファレンス・マニュアル』を参照してください。



## 例

サンプル・プログラム *ctos.c* には、ブラウザ・モード情報を処理するコードが記述されています。

## 機能

クライアントから発行可能な要求、および Open Server アプリケーションから返信可能な応答については、Open Server アプリケーションとクライアントがお互いに認識できる必要があります。たとえば、クライアントが言語要求を発行したくても、Open Server アプリケーションにはそのような要求を処理するパーサが提供されていない可能性があります。同様に、text データや image データをクライアントが処理できない場合は、クライアントは、Open Server アプリケーションからの text データや image データの送信を希望しない可能性があります。クライアント/サーバ接続の機能が、その接続で許可されるクライアントの要求とサーバ応答のタイプを決定します。

接続についてどの機能が有効であるかは、Open Server アプリケーションが最終的に決定します。有効と決定された機能がクライアントに受け入れられない場合は、接続をクローズするしかありません。

機能のネゴシエーションには、次の2つのタイプ、透過的および明示的があります。透過的ネゴシエーションの場合、Open Server アプリケーションは、可能なクライアント要求と Open Server 応答の「デフォルト・セット」を割り当てます。明示的ネゴシエーションの場合、Open Server アプリケーションには、`srv_capability_info` ルーチンを使って機能をネゴシエーションするためのコードが記述されます。

透過的ネゴシエーションは、Open Server と Open Client の双方のデフォルト動作の一部です。したがって、Open Server アプリケーションが機能のデフォルト・セット以外のものをサポートする場合は、`srv_capability_info` を呼び出す必要があります。

## 要求機能

表 2-1 は、要求機能の一覧を示します。

表 2-1 : 要求機能

CS_REQUEST 機能	意味	機能の対象
CS_CAP_EXTENDEDFAILOVER	拡張 HA フェールオーバー	接続
CS_CON_INBAND	バンド内 (非優先) アテンション	接続
CS_CON_OOB	バンド外 (優先) アテンション	接続
CS_CSR_ABS	指定された絶対カーソル・ローのフェッチ	カーソル
CS_CSR_FIRST	先頭カーソル・ローのフェッチ	カーソル
CS_CSR_LAST	最終カーソル・ローのフェッチ	カーソル
CS_CSR_MULTI	複数ロー・カーソルのフェッチ	カーソル
CS_CSR_PREV	前のカーソル・ローのフェッチ	カーソル
CS_CSR_REL	指定された相対カーソル・ローのフェッチ	カーソル
CS_DATA_BIGDATETIME	bigdatetime データ型	データ型
CS_DATA_BIGTIME	bigtime データ型	データ型
CS_DATA_BIN	binary データ型	データ型
CS_DATA_VBIN	可変長 binary データ型	データ型
CS_DATA_LBIN	long 可変長 binary データ型	データ型
CS_DATA_BIT	bit データ型	データ型
CS_DATA_BITN	NULL が許可される bit データ型	データ型
CS_DATA_BOUNDARY	boundary データ型	データ型
CS_DATA_CHAR	character データ型	データ型
CS_DATA_VCHAR	可変長 character データ型	データ型
CS_DATA_LCHAR	long 可変長 character データ型	データ型
CS_DATA_DATE	date データ型	データ型
CS_DATA_DATE4	short datetime データ型	データ型
CS_DATA_DATE8	datetime データ型	データ型
CS_DATA_DATETIMEN	NULL が許可される datetime 値	データ型
CS_DATA_DEC	decimal データ型	データ型
CS_DATA_FLT4	4 バイト float データ型	データ型
CS_DATA_FLT8	8 バイト float データ型	データ型
CS_DATA_FLTN	NULL が許可される float データ型	データ型
CS_DATA_IMAGE	image データ型	データ型
CS_DATA_INT1	tinyint データ型	データ型
CS_DATA_INT2	small integer データ型	データ型
CS_DATA_INT4	integer データ型	データ型

CS_REQUEST 機能	意味	機能の対象
CS_DATA_INT8	big integer データ型	データ型
CS_DATA_INTN	NULL integer	データ型
CS_DATA_MNY4	short money データ型	データ型
CS_DATA_MNY8	money データ型	データ型
CS_DATA_MONEYN	NULL money 値	データ型
CS_DATA_NUM	numeric データ型	データ型
CS_DATA_SENSITIVITY	sensitivity データ型	データ型
CS_DATA_TEXT	text データ型	データ型
CS_DATA_TIME	time データ型	データ型
CS_DATA_UCHAR	2 バイト character データ型	データ型
CS_DATA_UNITEXT	Unitext データ型	データ型
CS_DATA_XML	XML データ型	データ型
CS_OPTION_GET	現在のオプション値	データ型
CS_PROTO_DYNAMIC	TDS DESCIN/OUT プロトコルの使用	コマンド
CS_PROTO_DYNPROC	動的準備の前に "create proc" を追加	コマンド
CS_REQ_BCP	バルク・コピー要求	コマンド
CS_REQ_CURSOR	カーソル要求	コマンド
CS_REQ_DBRPC2	長い RPC 名の要求	コマンド
CS_REQ_DYN	動的 SQL 要求	コマンド
CS_REQ_LANG	言語要求	コマンド
CS_REQ_LARGEIDENT	長い識別子の要求	コマンド
CS_REQ_MIGRATE	マイグレーション要求	接続
CS_REQ_MSG	メッセージ・データ	コマンド
CS_REQ_MSTMT	1 つの Client-Library 要求に対して複数のサーバ・コマンド	接続
CS_REQ_NOTIF	イベント通知	接続
CS_REQ_SRPKTSIZE	サーバで指定された packetsize	接続
CS_REQ_PARAM	パラメータ・データ	コマンド
CS_REQ_RPC	リモート・プロシージャ要求	コマンド
CS_REQ_URGNOTIF	5.0 イベント通知プロトコルの使用	コマンド
CS_WIDETABLES	表ごとのカラムの拡大、数の増加	コマンド

## 応答機能

表 2-2 は、応答機能の一覧を示します。

---

**注意** 応答機能は、クライアントが受信したくない応答の種類を指定します。

---

表 2-2 : 応答機能

CS_RESPONSE 機能	意味	機能 対象
CS_CON_NOINBAND	バンド内 (非優先) アテンションなし	接続
CS_CON_NOOQB	バンド外 (優先) アテンションなし	接続
CS_DATA_NOBIGDATETIME	bigdatetime データ型なし	データ型
CS_DATA_NOBIGTIME	bigtime データ型なし	データ型
CS_DATA_NOBIN	binary データ型なし	データ型
CS_DATA_NOVBIN	可変長 binary データ型なし	データ型
CS_DATA_NOLBIN	long 可変長 binary データ型なし	データ型
CS_DATA_NOBIT	bit データ型なし	データ型
CS_DATA_NOBOUNDARY	boundary データ型なし	データ型
CS_DATA_NOCHAR	character データ型なし	データ型
CS_DATA_NOVCHAR	可変長 character データ型なし	データ型
CS_DATA_NOLCHAR	long 可変長 character データ型なし	データ型
CS_DATA_NODATE	date データ型なし	データ型
CS_DATA_NODATE4	short datetime データ型なし	データ型
CS_DATA_NODATE8	datetime データ型なし	データ型
CS_DATA_NODATETIMEN	NULL datetime 値なし	データ型
CS_DATA_NODEC	decimal データ型なし	データ型
CS_DATA_NOFLT4	4 バイト float データ型なし	データ型
CS_DATA_NOFLT8	8 バイト float データ型なし	データ型
CS_DATA_NOIMAGE	image データ型なし	データ型
CS_DATA_NOINT1	tinyint データ型なし	データ型
CS_DATA_NOINT2	small integer データ型なし	データ型
CS_DATA_NOINT4	integer データ型なし	データ型
CS_DATA_NOINT8	big integer データ型なし	データ型
CS_DATA_NOINTN	NULL integer なし	データ型
CS_DATA_NOMNY4	short money データ型なし	データ型
CS_DATA_NOMNY8	money データ型なし	データ型
CS_DATA_NOMONEYN	NULL money 値なし	データ型
CS_DATA_NONUM	numeric データ型なし	データ型
CS_DATA_NOSENSITIVITY	sensitivity データ型なし	データ型
CS_DATA_NOTEXT	text データ型なし	データ型
CS_DATA_NOTIME	time データ型なし	データ型

CS_RESPONSE 機能	意味	機能 対象
CS_DATA_NOUCHAR	2 バイト character データ型なし	データ型
CS_DATA_NOUNITEXT	unitext データ型なし	データ型
CS_DATA_NOXML	XML データ型なし	データ型
CS_NO_SRVPKTSIZE	サーバで指定された packetsize なし	接続
CS_RES_NOEED	拡張エラー結果なし	結果
CS_RES_NOMSG	メッセージ結果なし	結果
CS_RES_NOPARAM	結果パラメータなし	結果
CS_RES_NOTDSDEBUG	TDS デバッグ・トークンなし	結果
CS_RES_NOXNLMETADATA	テーブルのメタデータなし	結果
CS_NO_LARGEIDENT	長い識別子なし	コマンド
CS_NOWIDETABLES	表ごとのカラムサイズの拡大、カラム数の増加なし	コマンド

**注意** Open Server アプリケーションが、`srv_descfmt` ルーチンを使用してクライアントのデータ・フォーマットを定義するときに、Open Server は、関連するデータ型の応答機能が設定されていないことを確認します。その応答機能が設定されている場合は、クライアントはサーバがそのデータ型に関係する結果を送らないことを要求しているか、またはクライアント接続の TDS バージョンがそのデータ型をサポートしていないかのどちらかです。その場合、Open Server はエラーを起こし、`srv_descfmt` は `CS_FAIL` を返します。

## 透過的ネゴシエーション

Open Server には、機能のデフォルト値が設定されています。デフォルト値のリストについては、「[サーバワイドのデフォルト](#)」(31 ページ)を参照してください。これらのデフォルトはサーバ全体にわたるものであり、すべてのクライアント接続に適用されます。デフォルトが使用されると、Open Server がサポートする機能がすべてオンになります。

初期化中にサーバワイドなデフォルト値を変更するには、Open Server アプリケーションで `srv_props` ルーチンを呼び出します。「[srv\\_props](#)」(377 ページ)を参照してください。

DB-Library クライアントまたは Client-Library クライアントが Open Server アプリケーションにログインするときには、そのクライアントは希望する機能のリストをログイン・レコードの中に入れて送信します。透過的ネゴシエーションにおいては、Open Server はデフォルト値とクライアント値の共通部分を見つけます。その結果生じる値が、その接続でサポートされる値となります。

### 透過的ネゴシエーションが発生する場合

透過的ネゴシエーションは、次の場合に発生します。

- Open Server アプリケーションに、デフォルト・ハンドラ以外の SRV\_CONNECT ハンドラがない場合。
- Open Server アプリケーションのカスタムの SRV\_CONNECT イベント・ハンドラ内に、デフォルト機能を無効にするようなコードが明示的に記述されていない場合。

---

**注意** パススルー・モードでは、`srv_getloginfo` と `srv_setloginfo` によって機能ネゴシエーションが透過的に処理されます。

---

### サーバワイドのデフォルト

表 2-3 は、TDS のバージョン別に各要求機能のデフォルト設定をまとめたものです。1 は、TDS バージョンで機能がサポートされていることを示します。0 は、機能がサポートされていないことを示します。

表 2-3 : TDS バージョン別の要求機能

CS_REQUEST 機能	4.0	4.0.2	4.2	4.6	5.0
CS_CAP_EXTENDEDFAILOVER	0	0	0	0	1
CS_CON_INBAND	0	0	0	0	1
CS_CON_OOB	1	1	1	1	0
CS_CSR_ABS	0	0	0	0	0
CS_CSR_FIRST	0	0	0	0	0
CS_CSR_LAST	0	0	0	0	0
CS_CSR_MULTI	0	0	0	0	0
CS_CSR_PREV	0	0	0	0	0
CS_CSR_REL	0	0	0	0	0
CS_DATA_BIGDATETIME	0	0	0	0	1
CS_DATA_BIGTIME	0	0	0	0	1
CS_DATA_BIN	1	1	1	1	1
CS_DATA_BIT	1	1	1	1	1
CS_DATA_BITN	0	0	0	0	0
CS_DATA_BOUNDARY	0	0	0	0	0
CS_DATA_CHAR	1	1	1	1	1
CS_DATA_DATE	0	0	0	0	1
CS_DATA_DATE4	0	0	1	1	1
CS_DATA_DATE8	1	1	1	1	1
CS_DATA_DATETIME	1	1	1	1	1
CS_DATA_DEC	0	0	0	0	0
CS_DATA_FLT4	0	0	1	1	1
CS_DATA_FLT8	1	1	1	1	1
CS_DATA_FLTN	1	1	1	1	1
CS_DATA_IMAGE	1	1	1	1	1
CS_DATA_INT1	1	1	1	1	1
CS_DATA_INT2	1	1	1	1	1
CS_DATA_INT4	1	1	1	1	1
CS_DATA_INT8	0	0	0	0	1
CS_DATA_INTN	1	1	1	1	1
CS_DATA_LBIN	0	0	0	0	0
CS_DATA_LCHAR	0	0	0	0	0
CS_DATA_MNY4	0	0	1	1	1
CS_DATA_MNY8	1	1	1	1	1
CS_DATA_MONEYN	1	1	1	1	1
CS_DATA_NUM	0	0	0	0	0



<b>CS_REQUEST 機能</b>	<b>4.0</b>	<b>4.0.2</b>	<b>4.2</b>	<b>4.6</b>	<b>5.0</b>
CS_DATA_SENSITIVITY	0	0	0	0	0
CS_DATA_TEXT	1	1	1	1	1
CS_DATA_TIME	0	0	0	0	1
CS_DATA_UCHAR	0	0	0	0	1
CS_DATA_UNITEXT	0	0	0	0	1
CS_DATA_VBIN	1	1	1	1	1
CS_DATA_VCHAR	1	1	1	1	1
CS_DATA_XML	0	0	0	0	1
CS_OPTION_GET	0	0	0	0	0
CS_PROTO_DYNAMIC	0	0	0	0	0
CS_PROTO_DYNPROC	0	0	0	0	0
CS_REQ_BCP	1	1	1	1	1
CS_REQ_CURSOR	0	0	0	0	0
CS_REQ_DBRPC2	0	0	0	0	1
CS_REQ_DYN	0	0	0	0	0
CS_REQ_LANG	1	1	1	1	1
CS_REQ_LARGEIDENT	0	0	0	0	1
CS_REQ_MIGRATE	0	0	0	0	1
CS_REQ_MSG	0	0	0	0	0
CS_REQ_MSTMT	0	0	0	0	0
CS_REQ_NOTIF	0	0	0	1	1
CS_REQ_PARAM	0	0	0	0	0
CS_REQ_RPC	1	1	1	1	1
CS_REQ_SRPKTSIZE	0	0	0	0	1
CS_REQ_URGNOTIF	0	0	0	0	0
CS_WIDETABLES	0	0	0	0	1

表 2-4 は、TDS のバージョン別の各応答機能のデフォルト設定をまとめたものです。

- 1 は、TDS バージョンで機能がサポートされていないことを示します。
- 0 は、機能がサポートされていることを示します。

表 2-4 : TDS バージョン別の応答機能

CS_RESPONSE 機能	4.0	4.0.2	4.2	4.6	5.0
CS_CON_NOINBAND	1	1	1	1	1
CS_CON_NOOOB	0	0	0	0	0
CS_DATA_NOBIGDATETIME	1	1	1	1	0
CS_DATA_NOBIGTIME	1	1	1	1	0
CS_DATA_NOBIN	0	0	0	0	0
CS_DATA_NOBIT	0	0	0	0	0
CS_DATA_NOBOUNDARY	1	1	1	1	1
CS_DATA_NOCHAR	0	0	0	0	0
CS_DATA_NODATE4	1	1	0	0	0
CS_DATA_NODATE8	0	0	0	0	0
CS_DATA_NODATETIME	0	0	0	0	0
CS_DATA_NODEC	1	1	1	1	1
CS_DATA_NOFLT4	1	1	0	0	0
CS_DATA_NOFLT8	0	0	0	0	0
CS_DATA_NOIMAGE	0	0	0	0	0
CS_DATA_NOINT1	0	0	0	0	0
CS_DATA_NOINT2	0	0	0	0	0
CS_DATA_NOINT4	0	0	0	0	0
CS_DATA_NOINT8	1	1	1	1	0
CS_DATA_NOINTN	0	0	0	0	0
CS_DATA_NOLBIN	1	1	1	1	1
CS_DATA_NOLCHAR	1	1	1	1	1
CS_DATA_NOMNY4	1	1	0	0	0
CS_DATA_NOMNY8	0	0	0	0	0
CS_DATA_NOMONEY	0	0	0	0	0
CS_DATA_NONUM	1	1	1	1	1
CS_DATA_NOSENSITIVITY	1	1	1	1	1
CS_DATA_NOSINT1	1	1	1	1	0
CS_DATA_NOTEXT	0	0	0	0	0
CS_DATA_NOUCHAR	1	1	1	1	0
CS_DATA_NOUNITEXT	1	1	1	1	0
CS_DATA_NOVBIN	0	0	0	0	0
CS_DATA_NOVCHAR	0	0	0	0	0
CS_DATA_NOXML	1	1	1	1	0
CS_RES_NOEED	1	1	1	1	1
CS_RES_NOMSG	1	1	1	1	1

CS_RESPONSE 機能	4.0	4.0.2	4.2	4.6	5.0
CS_RES_NOPARAM	1	1	1	1	1
CS_RES_NOTDSDEBUG	1	1	1	1	1
CS_RES_NOXNLMETADATA	1	1	1	1	0
CS_NO_LARGEIDENT	1	1	1	1	0
CS_NO_SRPKTSIZE	1	1	1	1	0
CS_NOWIDETABLES	1	1	1	1	0

## 明示的ネゴシエーション

明示的ネゴシエーションは、接続時に `SRV_CONNECT` イベント・ハンドラ内部から開始されます。Open Server アプリケーションは、クライアントが送信する要求機能のリストを取得して、受け取る要求機能のリストを返します。今度は、クライアントが受け取りたくない応答機能、または、Open Server アプリケーションが返すことのできない応答機能を返すプロセスが繰り返されます。

アプリケーションは、一度に1つずつ機能を取得して送信したり、機能を取得して機能のビットマスク全体をただちに送信したりすることができます。Open Server 提供のマクロを使用して、機能マスク内のビットをテスト、クリア、設定できます。[「機能マクロ」\(36 ページ\)](#)を参照してください。

## 一度に1つずつの機能のネゴシエーション

要求機能を一度に1つずつネゴシエートするには、ネゴシエートする機能ごとに次の呼び出しを行う必要があります。

- 1 `cmd` 引数を `CS_GET` に、`type` 引数を `CS_CAP_REQUEST` に、`capability` 引数を目的の機能に設定して `srv_capability_info` を呼び出します。`*valp` 引数に `CS_TRUE` が含まれている場合は、クライアントは、このようなタイプの機能を要求します。`*valp` 引数に `CS_FALSE` が含まれている場合は、クライアントはこのような機能を要求しません。
- 2 `cmd` 引数を `CS_SET` に、`type` 引数を `CS_CAP_REQUEST` に、`capability` 引数を目的の機能に、`*valp` をブール値に設定して、`srv_capability_info` を呼び出します。アプリケーションがこのタイプの機能をサポートする場合は `*valp` を `CS_TRUE` に設定し、サポートしない場合は `CS_FALSE` に設定します。

アプリケーションは、`type` 引数を `CS_CAP_RESPONSE` に設定しなければならないこと以外は、同様の方法で応答機能をネゴシエートします。

Open Server アプリケーションは、明示的にネゴシエートする要求および応答機能についてだけ `srv_capability_info` を呼び出す必要があります。その他の機能には、すべてデフォルト値が使用されます。

## 機能ビットマスクを使用したネゴシエーション

機能ビットマスクを使用して要求機能をネゴシエートするには、アプリケーションは次の処理が必要です。

- 1 `cmd` 引数を `CS_GET` に、`type` 引数を `CS_CAP_REQUEST` に、`capability` 引数を `CS_ALL_CAPS` に設定し、`valp` がビットマスクが含まれる `CS_CAP_TYPE` 構造体を指すように設定して `srv_capability_info` を呼び出すことにより、ビットマスク全体を読み込みます。
- 2 ビットマスク内の特定のビットをテスト、設定、またはクリアするには、`CS_TST_CAPMASK`、`CS_SET_CAPMASK`、`CS_CLR_CAPMASK` の各マクロを使用します。

アプリケーションは、`type` 引数を `CS_CAP_RESPONSE` に設定しなければならないこと以外は、同様の方法で応答機能をネゴシエートします。

ゲートウェイ・アプリケーションは、機能のネゴシエーションを行うためには、マスク方式を使用します。次の図に示されているように、ゲートウェイは、`srv_capability_info` を呼び出してリモート・クライアントの機能マスクを取得し、`ct_capability` を呼び出してそれらの機能をリモート・サーバに送信してから、`ct_connect` を呼び出します。リモート接続が確立すると、ゲートウェイは、リモート・サーバが `ct_capability` を使用して送信した機能マスクを取得し、`srv_capability_info` を使用して、リモート・クライアントの接続時にそれらの機能マスクを定義できます。

## 機能マクロ

表 2-5 は、アプリケーションが機能ビットマスクを操作するために使用するマクロについて説明しています。

表 2-5 : 機能マクロ

マクロ名	関数
CS_TST_CAPMASK	特定の機能が CS_TRUE と CS_FALSE のどちらに設定されているかをテストする。
CS_SET_CAPMASK	特定の機能を CS_TRUE に設定する。
CS_CLR_CAPMASK	特定の機能を CS_FALSE に設定する。

機能を明示的に扱う場合、デフォルト設定を使用する代わりに、次の2つの規則を適用します。

- CS\_CAP\_REQUEST**  
 アプリケーションでは、CS\_CAP\_REQUEST 機能を "on" ステータスから "off" ステータスに切り換える操作だけが可能です。  
  
 アプリケーションが CS\_CAP\_REQUEST 機能を "off" にしようとしたときに、すでに "off" ステータスになっている場合、Open Server はデフォルトのステータスをリストアして、エラーが発生しないようにします。
- CS\_CAP\_RESPONSE**  
 アプリケーションでは、CS\_CAP\_RESPONSE 機能を "off" ステータスから "on" ステータスに切り換える操作だけが可能です。  
  
 アプリケーションが CS\_CAP\_RESPONSE 機能を "on" にしようとしたときに、すでに "on" ステータスになっている場合、Open Server はデフォルトのステータスをリストアして、エラーが発生しないようにします。

## 機能情報の特定の検索

Open Server アプリケーションは、特定のクライアント接続に対して有効な機能リストを取得するために、いつでも、どのハンドラの内部からでも `srv_capability_info` を呼び出すことができます。しかし、SRV\_CONNECT イベント・ハンドラにおいては、取得された機能マスクはその接続の最終的なマスクとはならないことに注意してください。これらのマスクは、Open Server アプリケーションのデフォルトと結合したクライアントが要求した機能のものです。SRV\_CONNECT ハンドラが返されるまで、接続機能が最終のものとはなりません。

## バージョン 10.0 以前のクライアントに関する注意

Open Server アプリケーションは、どのような TDS バージョンを実行しているクライアントとでも機能をネゴシエートできます。10.0 以前のクライアントを接続する場合、Open Server は、機能ネゴシエーションのシミュレートを行います。この場合に、Open Server アプリケーションは、クライアントが実行している TDS バージョンを認識する必要はありません。

### 例

オンラインのサンプル・プログラム *ctos.c* には、機能ネゴシエーションのコードが記述されています。

## クライアント・コマンド・エラー

クライアントは、不完全または意味のない要求を Open Server アプリケーションに送ることがあります。クライアントのコードに誤りがあったりネットワークに問題があったりすると、要求は不完全になったり無意味になったりします。Open Server アプリケーションは、クライアント要求に対するイベント・ハンドラで、適切なエラー・メッセージをクライアントに送信して、これらのエラーを処理します。

### *srv\_sendinfo*

によるメッセージの送信

Open Server アプリケーションは *srv\_sendinfo* を呼び出し、エラー・メッセージをクライアントに送ります。Open Server アプリケーションは、*CS\_SERVERMSG* 構造体にメッセージを記述し、*srv\_sendinfo* を呼び出して、この記述をクライアントに送ります。

[「CS\\_SERVERMSG 構造体」\(64 ページ\)](#) を参照してください。

## 長いメッセージの連続化

Open Server アプリケーションは、メッセージ・テキストを CS\_SERVERMSG 構造体の text フィールドに格納します。text の最大長は CS\_MAX\_MSG バイトです。

Open Server アプリケーションは、メッセージ・テキスト全体を送るために必要な数だけ CS\_SERVERMSG 構造体を使用します。アプリケーションは最初の構造体内の CS\_MAX\_MSG バイトだけ返し、2 番目の構造体内の 2 番目の CS\_MAX\_MSG バイトを返します。以下同様にこの処理を繰り返します。この処理はメッセージの「連続化」と呼ばれます。

アプリケーションは、「連続化した」数と同じ数だけ srv\_sendinfo を呼び出します。メッセージ全体が 1 つの構造体に収まる場合、アプリケーションは srv\_sendinfo を一度だけ呼び出します。

## 連続化されたメッセージに対する CS\_SERVERMSG 構造体のフィールド

CS\_SERVERMSG 構造体の status フィールドは、構造体がメッセージ全体を含むのかメッセージのまとまり (部分) を含むのかを示します。

表 2-6 は、連続化されたメッセージに関連する status 値のリストです。

表 2-6 : 連続化されたメッセージの status 値

記号値	意味
CS_FIRST_CHUNK	メッセージ・テキストは、メッセージの最初の部分。
CS_LAST_CHUNK	メッセージ・テキストは、メッセージの最後の部分。 構造体の中のメッセージ・テキストがメッセージ全体の場合、アプリケーションは CS_FIRST_CHUNK と CS_LAST_CHUNK の両方を設定する。 構造体の中のメッセージ・テキストが中間のテキストの場合、アプリケーションは CS_FIRST_CHUNK と CS_LAST_CHUNK のどちらも設定しない。

CS\_SERVERMSG 構造体の中の textlen フィールドは、常に現在のメッセージのまとまりの長さを反映します。

CS\_SERVERMSG 構造体の他のフィールドは、各メッセージのまとまりごとに繰り返されます。

## 拡張エラー・データ

サーバ・メッセージは、関連する「拡張エラー・データ」を含んでいる場合があります。拡張エラー・データは、エラーに関する追加情報です。

Adaptive Server Enterprise メッセージの場合、通常、追加情報にはエラーを引き起こしたカラムが記述されています。

## 拡張エラー・データの利点

ユーザがデータを入力または編集できるクライアント・アプリケーションは、カラム・レベルでユーザにエラーをレポートする必要があります。しかし、標準サーバ・メッセージ・メカニズムでは、カラム・レベルの情報はサーバ・メッセージのテキスト内でしか使用できません。拡張エラー・データは、カラム・レベルの情報に簡単にアクセスするための方法をアプリケーションに提供します。

たとえば、ユーザが pubs2 データベースの titleauthor テーブルにデータを入力および編集できるクライアント・アプリケーションがあるとします。titleauthor は、2つのカラム au\_id と title\_id からなる「キー」を使用します。既存のローに合致する au\_id と title\_id を含むローを入力しようとする、「重複キー」メッセージがクライアント・アプリケーションに送信されます。

このメッセージを受信したクライアント・アプリケーションは、ユーザが修正できるよう、問題のカラムをユーザに指示する必要があります。この情報は、メッセージ・テキスト以外の、重複キー・メッセージでは使用できません。この情報は、拡張エラー・データとして使用できます。

## 拡張エラー・データのクライアントへの送信

メッセージで拡張エラー・データが使用可能な場合、Open Server アプリケーションは CS\_HASEED 構造体の status フィールドを CS\_SERVERMSG に設定します。

Open Server アプリケーションは、srv\_sendinfo ルーチンのパラメータとして拡張エラー・データを送ります。アプリケーションは、srv\_descfmt、srv\_bind、srv\_xferdata を使用してエラー・パラメータの記述、バインド、送信を行います。



アプリケーションは、`srv_sendinfo` を呼び出した直後に、エラー・パラメータを記述、バインド、送信してから、他の結果を返したり `srv_senddone` を呼び出したりします。アプリケーションは、`type` 引数を `SRV_ERRORDATA` に設定して、`srv_descfmt`、`srv_bind`、`srv_xferdata` を呼び出さなければなりません。

アプリケーションが `CS_SERVERMSG` 構造体の `status` フィールドを `CS_HASEED` に設定して `srv_sendinfo` を呼び出したがエラー・パラメータの送信に失敗した場合、`Open Server` は、アプリケーションが `srv_senddone` を呼び出したときに致命的なプロセス・エラーを起こします。

## 接続マイグレーション

接続マイグレーションにより、`Open Server` アプリケーションはその負荷を動的に分散し、透過的フェールオーバをサポートできます。また、異なる関数を実行する複数の `Open Server` アプリケーションがある場合、クライアントの要求を満たすことができる `Open Server` にクライアントをリダイレクトできます。

以下で説明するアプリケーション・プログラミング・インタフェース (API) を使用すると、`Open Server` はマイグレーション要求を開始、完了、およびキャンセルし、クライアントからのマイグレーション・メッセージに反応することができます。さらに、新しい接続がマイグレーション接続であるかどうかを検出して、接続からユニークな識別子を取得することもできます。

## バッチ内マイグレーションとアイドル・マイグレーション

バッチ内マイグレーションでは、クライアントはオリジナル・サーバから結果を待機中にマイグレートします。反対に、アイドル・マイグレーションでは、クライアントはオリジナル・サーバから結果を待機しません。

バッチ内マイグレーションにより、Open Server は接続のマイグレーション後まで、結果の送信または完了を遅延できるようになります。これは、Open Server が特定の要求を処理できない場合や、要求を完了する時間がない場合に役立ちます。バッチ内マイグレーションでは、Open Server はオリジナル・サーバーからの結果の一部を送信でき、マイグレーション後にクライアントのマイグレーション先のサーバが SRV\_MIGRATE\_RESUME イベント・ハンドラからの結果の残りを送信できます。

---

**注意** 新しいサーバが結果をまったく送信しない場合、オリジナル・サーバが結果全体をクライアントに送信できます。同様に、新しいサーバが結果全体をクライアントに送信する必要がある場合、オリジナル・サーバが結果をクライアントにまったく送信しないことがあります。

---

バッチ内マイグレーションでは、未送信のコマンドおよびメッセージがクライアント・コンテキストの一部であることをアプリケーションが確認する必要があります。新しいサーバは、コマンドの影響を受けるローの数と、接続のトランザクション・ステータスにアクセスする必要もあります。新しいサーバは、`srv_senddone()` を使用してこの情報をクライアントに送信します。

## コンテキスト・マイグレーション

Open Server は、クライアントの接続のシームレスなマイグレーションをサポートします。ただし、クライアントのコンテキストの共有およびマイグレーションはアプリケーションが行います。共有ファイルの使用や、ネットワーク通信など、コンテキスト・マイグレーションはさまざまな方法で実装できます。

バッチ内マイグレーションの場合、クライアントのマイグレーション先のサーバはオリジナル・サーバ内で発生したイベントのタイプを認識しません。アプリケーションがこの情報を必要とする場合、クライアントのコンテキストの一部として情報をマイグレートする必要があります。

アイドル・マイグレーションでは、クライアントは Open Server からの実際の結果を待機しません。アクティブなクエリがないため、アイドル・マイグレーションはバッチ内マイグレーションより簡単に実装できます。ただし、アイドル・マイグレーションでも、クライアントがマイグレーションを開始する前に発生した保留中の要求をすべてアプリケーションが満たす必要があります。

## 接続マイグレーションで使用される API

この項では、接続マイグレーションをサポートする API について説明します。「[クライアントに対する異なるサーバへのマイグレーションの指示](#)」(50 ページ)を参照してください。

### CS\_REQ\_MIGRATE

CS\_REQ\_MIGRATE 要求機能は、クライアントがマイグレーション・プロトコルをサポートするかどうかと、クライアントが要求時に別のサーバへのマイグレーションを実行できるかどうかを指定します。srvcapability\_info() を使用すると、CS\_REQ\_MIGRATE 機能情報を取得できます。次に例を示します。

```
CS_RETCODE    ret;
CS_BOOL       migratable;
ret = srvcapability_info(sp, CS_GET, CS_CAP_REQUEST,
                        CS_REQ_MIGRATE, &migratable);
```

### SRV\_CTL\_MIGRATE

SRV\_CTL\_MIGRATE は、srvcctlinfo() 制御タイプです。クライアントがマイグレーションをサポートし、セッションへの最初の接続時にセッション ID を受信した場合、SRV\_CTL\_MIGRATE を使用すると、マイグレーション要求をクライアントに送信したり、以前のマイグレーション要求をキャンセルすることができます。

#### クライアント・マイグレーションの要求

次のサンプル・コードは、サーバ "target" にマイグレートする要求をクライアントに送信します。

```
CS_RETCODE    ret;
SRV_CTLITEM  *srvcitems;
CS_CHAR       *target;
```

```

/*
** request a migration to server 'target'
*/
srvitems = (SRV_CTLITEM *) srv_alloc(sizeof
    (SRV_CTLITEM));
srvitems[0].srv_ctlitemtype = SRV_CT_SERVERNAME;
srvitems[0].srv_ctllength = strlen(target);
srvitems[0].srv_ctlptr = target;
ret = srv_send_ctlinfo(sp, SRV_CTL_MIGRATE, 1,
    srvitems);
srv_free(srvitems);

```

アプリケーションは、マイグレーションがすでに要求されている場合でも、`SRV_CTL_MIGRATE` 制御タイプを送信できます。Open Server は、以前のマイグレーション要求をキャンセルし、新しい要求をクライアントに送信します。新しいマイグレーション要求の戻り値は、次のとおりです。

戻り値	説明
CS_SUCCEED	マイグレーション要求が正常に送信されました。
CS_FAIL	次のいずれかの理由のため、マイグレーション要求に失敗しました。 <ul style="list-style-type: none"> <li>• Open Server スレッドが接続マイグレーションをサポートしていない。</li> <li>• 以前のマイグレーション要求が送信され、クライアントが新しいサーバへのマイグレーションを開始した。</li> </ul>

#### マイグレーションのキャンセル

`SRV_CTL_MIGRATE` 制御タイプを使用して、以前のマイグレーション要求をキャンセルすることもできます。この場合、*paramcnt* を 0 にし、*param* を NULL ポインタにする必要があります。次に例を示します。

```

ret = srv_send_ctlinfo(sp, SRV_CTL_MIGRATE, 0, NULL);
if (ret != CS_SUCCEED)
{
    ...
}

```

`SRV_CTL_MIGRATE` は、Open Server アプリケーション内のどのスレッドでも使用できます。ただし、クライアント・スレッドの接続のマイグレーションをキャンセルするスレッドの要件は、自身の接続マイグレーションをキャンセルするクライアント・スレッドの要件とは異なります。

- どの Open Server スレッドでもマイグレーションをキャンセルできますが、SRV\_MIGRATE\_STATE イベント・ハンドラが、クライアントでマイグレーションの準備ができたことをクライアント・スレッドに知らせる前に、キャンセルが要求される必要があります。
- クライアント・スレッドは、SRV\_MIGRATE\_STATE イベント・ハンドラ内でもマイグレーションをキャンセルできます。ただし、クライアント・スレッドは、SRV\_MIG\_READY ステータスの SRV\_MIGRATE\_STATE イベントを終了した後は、マイグレーションをキャンセルできません。

マイグレーションのキャンセルの戻り値は次のとおりです。

戻り値	説明
CS_SUCCEED	マイグレーション要求が正常にキャンセルされました。
CS_FAIL	次のいずれかの理由のため、マイグレーションのキャンセルに失敗しました。 <ul style="list-style-type: none"> <li>• 進行中のマイグレーションがない。</li> <li>• クライアントが新しいサーバへのマイグレーションを開始した。</li> </ul>

**注意** マイグレーション要求が正常にキャンセルされると、Open Server は新しいマイグレーション・ステータス・イベントをトリガしません。

## SRV\_MIGRATE\_RESUME

クライアントが結果の待機中に新しいサーバにマイグレートすると、クライアント接続が正常にマイグレートされた後、新しいサーバは SRV\_MIGRATE\_RESUME イベントを呼び出します。マイグレーション要求が失敗した場合やキャンセルされた場合、イベントはオリジナル・サーバから呼び出されます。

SRV\_MIGRATE\_RESUME イベント・ハンドラでは、アプリケーションは実際の結果をクライアントに送信する必要はありません。ただし、必ず送信する必要がある SRV\_DONE\_FINAL 結果タイプは除きます。デフォルトの SRV\_MIGRATE\_RESUME がクライアントに送信する結果は、SRV\_DONE\_FINAL だけです。

SRV\_MIGRATE\_RESUME イベント・ハンドラの例を次に示します。

```
/*
** Simple migrate_resume event handler.
*/
```

```

CS_RETCODE CS_PUBLIC
migrate_resume_handler(SRV_PROC *sp)
{
    CS_RETCODE    ret;
    ret = srv_senddone(sp, SRV_DONE_FINAL,
        CS_TRAN_COMPLETED, 0);
    if (ret == CS_FAIL)
    {
        ...
    }
    return CS_SUCCEED;
}

...

/*
** Install the migrate-resume event handler
*/
srv_handle(server, SRV_MIGRATE_RESUME,
    migrate_resume_handler);
...

```

## SRV\_MIGRATE\_STATE

SRV\_MIGRATE\_STATE は、マイグレーション・ステータスが SRV\_MIG\_READY または SRV\_MIG\_FAILED に移行すると必ずトリガされるイベントです。移行は、クライアントからマイグレーション・メッセージが送信された結果です。SRV\_MIGRATE\_STATE イベント・ハンドラは、次のような状況で呼び出されます。

SRV_T_MIGRATE_STATE	状況	発生する可能性があるアプリケーションの動作
SRV_MIG_READY	クライアントが、要求を検出し、マイグレーションの準備ができたことを示すメッセージをサーバに送信した。サーバは、マイグレーションを続行するかどうかを判断する。	次のいずれか1つ。 <ul style="list-style-type: none"> <li>コンテキストを他のサーバが使用できるようにする。</li> <li>マイグレーションがなくなるとアプリケーションが判断した場合は、マイグレーションをキャンセルする。</li> <li>新しいマイグレーション・ターゲットが選択されている場合は、別のマイグレーションを要求する。</li> </ul>

SRV_T_MIGRATE_STATE	状況	発生する可能性があるアプリケーションの動作
SRV_MIG_FAILED	クライアントが、マイグレーションが失敗したことを示すメッセージをサーバに送信した。	次のいずれか1つ。 <ul style="list-style-type: none"> <li>クライアント・コンテキストにアクセスし、接続の提供を続行する。</li> <li>別のマイグレーションを要求する。</li> </ul>

SRV\_MIGRATE\_STATE イベント・ハンドラの例を次に示します。

```

/*
** Simple migrate-state event handler
*/
CS_RETCODE CS_PUBLIC
migrate_state_handler(SRV_PROC *sp)
{
    SRV_MIG_STATE migration_state;
    ret = srv_thread_props(sp, CS_GET,
        SRV_T_MIGRATE_STATE, &migration_state,
        sizeof (migration_state), NULL);

    ...

    switch(migration_state)
    {
        case SRV_MIG_READY:
            ...
        case SRV_MIG_FAILED:
            ...
    }
}

...

/*
** Install the migrate-state change event handler
*/
srv_handle(server, SRV_MIGRATE_STATE,
    migrate_state_handler);
...

```

SRV\_MIGRATE\_STATE イベント・ハンドラを使用するときは、次のことが当てはまります。

- クライアント・スレッドが `SRV_MIGRATE_STATE` イベント・ハンドラ内からマイグレーションをキャンセルした場合、アプリケーションはコンテキストの一貫性を保つ必要があります。たとえば、アプリケーションが作成したコンテキストを別のサーバが使用することはありません。
- 新しいマイグレーション要求が `SRV_MIGRATE_STATE` イベント・ハンドラ内から送信された場合、このハンドラはクライアントが新しく要求されたマイグレーションを開始する準備ができたときに再度呼び出されます。

## SRV\_T\_MIGRATE\_STATE プロパティと SRV\_MIG\_STATE 列挙型

`SRV_T_MIGRATE_STATE` は、クライアントのマイグレーション・ステータスを示します。`SRV_T_MIGRATE_STATE` は、どのスレッドでもアクセスできる読み込み専用プロパティです。示されるマイグレーション・ステータスは次のとおりです。

状態	値	説明
<code>SRV_MIG_NONE</code>	0	進行中のマイグレーションがない。
<code>SRV_MIG_REQUESTED</code>	1	マイグレーションがサーバにより要求された。
<code>SRV_MIG_READY</code>	2	クライアントが要求を受信し、マイグレーションの準備ができた。
<code>SRV_MIG_MIGRATING</code>	3	クライアントが指定されたサーバにマイグレート中である。
<code>SRV_MIG_CANCELLED</code>	4	マイグレーション要求がキャンセルされた。
<code>SRV_MIG_FAILED</code>	5	クライアントがマイグレーションに失敗した。

`SRV_MIG_STATE` は、`SRV_T_MIGRATE_STATE` プロパティをモデル化する列挙データ型です。次のように `SRV_MIG_STATE` を宣言します。

```
typedef enum
{
    SRV_MIG_NONE,
    SRV_MIG_REQUESTED,
    SRV_MIG_READY,
    SRV_MIG_MIGRATING,
    SRV_MIG_CANCELLED,
    SRV_MIG_FAILED
} SRV_MIG_STATE;
```



次のサンプル・コードは、`SRV_T_MIGRATE_STATE` 値を取得する方法を示しています。マイグレーションに成功した場合、クライアントが終了し、`SRV_DISCONNECT` イベント・ハンドラが `SRV_MIG_MIGRATING` ステータスで呼び出されます。

```
CS_RETCODE    ret;
SRV_MIG_STATE migration_state;
ret = srv_thread_props(sp, CS_GET, SRV_T_MIGRATE_STATE,
    &migration_state, sizeof (migration_state), NULL);
if (ret != CS_SUCCEEDED)
{
    ...
}
```

## SRV\_T\_MIGRATED

`SRV_T_MIGRATED` は、接続が新しい接続であるか、マイグレートされた接続であるかを示す **Boolean** プロパティです。この読み込み専用プロパティは、クライアントがマイグレート中であるか、サーバにマイグレートした場合は `true` に設定されます。次のサンプル・コードは、`SRV_T_MIGRATED` の値を取得します。

```
CS_RETCODE    ret;
CS_BOOL       migrated;
status = srv_thread_props(sp, CS_GET, SRV_T_MIGRATED,
    &migrated, sizeof (migrated), NULL);
```

## SRV\_T\_SESSIONID

`SRV_T_SESSIONID` は、クライアントから Open Server に送信されたセッション ID を取得するスレッド・プロパティです。次の場合は、`srv_thread_props()` 関数を使用して `SRV_T_SESSIONID` プロパティを設定できます。

- `srv_thread_props(CS_SET, SRV_T_SESSIONID)` 呼び出しが `SRV_CONNECT` イベント・ハンドラ内で行われた。
- クライアントで接続マイグレーションまたは高可用性がサポートされている。

次のサンプル・コードは、`SRV_T_SESSIONID` プロパティを設定します。

```
CS_RETCODE    ret;
CS_SESSIONID  hasessionid;
ret = srv_thread_props(sp, CS_SET, SRV_T_SESSIONID,
```

```
hasessionid, sizeof(hasessionid), NULL);
```

---

**注意** HA フェールオーバーの場合、セッション ID がクライアントに送信されるように `srv_negotiate()` シーケンスをプログラミングする必要があります。

---

## クライアントに対する異なるサーバへのマイグレーションの指示

この項では、他のサーバにクライアントをマイグレートするための Open Server の要件について説明します。クライアントを異なるサーバにマイグレートするとき、アプリケーションは次の処理を実行する必要があります。

- 1 ユニーク・セッション ID を作成し、それを接続ハンドラでクライアントに送信します。
- 2 接続マイグレーションを開始します。
- 3 マイグレーション・イベントを処理します。
- 4 接続のセッション ID を使用して、他のサーバへの接続のコンテキストを共有します。
- 5 (オプション) 既存のハンドラで進行中のマイグレーションを操作します。

これ以降は、これらのアクティビティについて詳細に説明します。

## クライアントへのマイグレーションの要求

Open Server は、`srv_send_ctlinfo()` を使用してマイグレーション要求をクライアントに送信できます。クライアント・マイグレーションは、どの Open Server スレッドからでも要求できます。

## 接続 (SRV\_CONNECT) イベントのマイグレーション

SRV\_CONNECT イベント・ハンドラでは、アプリケーションで次の処理を実行する必要があります。

- `SRV_T_MIGRATED` プロパティを確認し、接続がマイグレートされた接続であるかどうかを判断します。マイグレートされた接続である場合、アプリケーションはクライアントにより提供されるセッション ID に基づいてコンテキストにアクセスする必要があります。セッション ID は、`SRV_T_SESSIONID` スレッド・プロパティを使用して取得できます。
- `CS_REQ_MIGRATE` を確認して、クライアントが接続マイグレーションをサポートするかどうかを判別します。クライアントが接続マイグレーションをサポートする場合、アプリケーションは `SRV_T_SESSIONID` プロパティを使用してセッション ID をクライアントに送信する必要があります(クライアントがまだセッション ID を取得していない場合)。クライアントにセッション ID を割り当てることで、アプリケーションは必要が生じたときにマイグレートするようにクライアントに指示できます。

## マイグレーション・ステータス (`SRV_MIGRATE_STATE`) イベントの管理

`SRV_MIGRATE_STATE` イベント・ハンドラは、マイグレーションの状態変更を管理し、変更ごとに適切なアクションを実行する必要があります。

- `SRV_MIG_READY` に変更された `SRV_MIGRATE_STATE`  
"ready" マイグレーション・ステータスは、クライアントでマイグレーションの準備ができており、現在のところは要求を送信していないことを示しています。`SRV_MIGRATE_STATE` イベント・ハンドラでは、`Open Server` はマイグレート先のサーバとクライアント・コンテキストを共有します。その後、アプリケーションはイベント・ハンドラから戻ることができ、`Open Server` はマイグレーションの開始を自動的にクライアントに指示できます。
- `SRV_MIG_FAILED` に変更された `SRV_MIGRATE_STATE`  
マイグレーション・ステータスが "failed" に変更されたために `SRV_MIGRATE_STATE` イベント・ハンドラがトリガされた場合、アプリケーションは再度コンテキストにアクセスする必要があります。アプリケーションで `srv_send_ctlinfo()` 関数を使用して `SRV_MIG_STATE` イベント・ハンドラから別のマイグレーション試行を要求できます。ただし、クライアントは再度マイグレートする準備ができたことを示す前に、別のクエリを送信した可能性があります。アプリケーションは、そのような要求を処理またはマイグレートできる必要があります。

## クライアント・コンテキストの共有

サーバがクライアントの処理を開始して続行する場合、サーバはクライアントのセッション ID により識別されるクライアントのコンテキストにアクセスできる必要があります。通常、クライアントのコンテキストには、クライアントのイベント・ハンドラがアクセス可能なデータ (グローバル・データなど) が含まれています。接続に必要なコンテキストの量は、Open Server アプリケーションが提供するサービスによって決まります。サービスのコンテキストが少なければ少ないほど、共有する必要があるコンテキストは少なくなります。

## マイグレーション再開 (SRV\_MIGRATE\_RESUME) イベントの管理

アプリケーションは、残りの結果とメッセージを SRV\_MIGRATE\_RESUME イベント・ハンドラ内でクライアントに送信します。Open Server がクライアントに送信する結果とメッセージは、アプリケーションとマイグレーションのタイプによって異なります。ただし、アプリケーションは SRV\_DONE\_FINAL 結果タイプをクライアントに送信して、SRV\_MIGRATE\_RESUME イベント・ハンドラを終了する必要があります。

## 切断 (SRV\_DISCONNECT) イベントのマイグレーション

SRV\_DISCONNECT イベント・ハンドラでは、アプリケーションで SRV\_T\_MIGRATE\_STATE を確認してクライアントのマイグレーション状態を判別する必要があります。

- マイグレーション・ステータス SRV\_MIG\_REQUESTED は、クライアントがマイグレーション要求に応答できるようになる前に Open Server アプリケーションが接続を終了したため、SRV\_DISCONNECT イベントがトリガされたことを示しています。
- マイグレーション・ステータス SRV\_MIG\_MIGRATING は、新しいサーバへのマイグレーションが成功した後でクライアント・アプリケーションが接続を閉じたため、SRV\_DISCONNECT イベントがトリガされたことを示しています。
- 他のすべてのマイグレーション状態の場合は他のサーバがこのコンテキストを取得しないため、クライアントは接続固有のコンテキストがクリーンアップされるようにする必要があります。

## バッチ内マイグレーションの管理

長時間実行されているイベント・ハンドラは、マイグレーション・ステータスを定期的に検査する必要があります。他の **Open Server** スレッドは、イベント・ハンドラ・プロセスがまだ実行中の間でも、マイグレーション要求を送信できます。この場合、イベント・ハンドラは可能であればプロセスを中断し、接続が新しいサーバにマイグレートされるまで結果の生成と送信を延期する必要があります。

## アテンション処理

クライアントがアテンション・メッセージを送信して未処理の要求をキャンセルすると、**SRV\_T\_GOTATTENTION** スレッド・プロパティが **CS\_TRUE** に設定されて **SRV\_ATTENTION** イベント・ハンドラが呼び出されます。接続マイグレーションで必要になる具体的なアテンション処理は次のとおりです。

- **SRV\_MIGRATE\_STATE** イベント・ハンドラと **SRV\_MIG\_READY** ステータスの場合

マイグレーションの準備ができたことをクライアントが示す前にアテンション・メッセージが **SRV\_MIGRATE\_STATE** イベント・ハンドラに到着した場合、**SRV\_MIGRATE\_STATE** イベント・ハンドラが終了すると **Open Server** はアテンションを確認します。これにより、クライアントからの要求が完了します。マイグレーションが成功すると、クライアントのマイグレート先のサーバはこのアテンション・メッセージを受信しません。クライアントが **Open Server** からの結果を待機していないため、**SRV\_MIGRATE\_RESUME** イベント・ハンドラは呼び出されません。

したがって、アプリケーションは他のサーバがコンテキストを使用できるようにする前に、**SRV\_T\_GOTATTENTION** プロパティが **CS\_TRUE** に設定されているかどうかを確認する必要があります。**SRV\_T\_GOTATTENTION** が **CS\_TRUE** に設定されている場合、コンテキストを更新して、クライアントが操作をキャンセルしたことを示す必要があります。

- **SRV\_MIGRATE\_RESUME** イベント・ハンドラの場合

マイグレーションの準備ができたことが示され、マイグレーションが成功した後、クライアントがアテンション・メッセージを送信した場合、アテンションはクライアントのマイグレート先のサーバに送信されます。このため、マイグレーションが成功すると、オリジナル・サーバがコンテキストを更新してキャンセルを反映した場合でも、SRV\_MIGRATE\_RESUME イベント・ハンドラがアテンションを受信可能な可能性があります。したがって、アプリケーションは SRV\_MIGRATE\_RESUME イベント・ハンドラを実行する前に、クライアントがアテンションをサーバに送信したかどうかを確認する必要があります。

### Open Server の切断

アプリケーションは、マイグレーションが要求されたときでもクライアント接続を終了できますが、Open Server が終了コマンドを発行する直前に送信された新しいクライアント・コマンドは失われる可能性があります。これを回避するには、アプリケーションが次の処理を実行する必要があります。

- 可能であれば、クライアントがマイグレーションを指示された場合は接続を終了しないようにします。
- クライアントを切断する必要がある場合、Open Server はマイグレーションを要求する前に適切な待機時間を設定する必要があります。これにより、別のコマンドを発行するまでにマイグレーション要求を検出する時間がクライアントに与えられます。
- Open Server が接続を終了すると、SRV\_DISCONNECT イベント・ハンドラが呼び出されます。このハンドラ内で、マイグレーション・ステータスが SRV\_MIG\_REQUESTED に設定されたままの場合は、他のサーバがコンテキストを使用できるようにします。

### マイグレートされたクライアントからの接続の受け入れ

Open Server は、SRV\_CONNECT イベント・ハンドラ内の SRV\_T\_MIGRATED プロパティを検査することで、新しい接続がマイグレート中か、マイグレートを完了したかを判断できます。SRV\_T\_MIGRATED が TRUE の場合、SRV\_T\_SESSIONID プロパティを使用してクライアントからセッション ID を取得できます。セッション ID を変更することもできますが、セッション ID は後でクライアントをマイグレートするために必要ではありません。

クライアントのマイグレート時にクライアントがコマンドを実行していた場合、SRV\_MIGRATE\_RESUME イベントがトリガされ、Open Server は結果をクライアントに送信してコマンドを完了できます。アプリケーションは、セッション情報の取得を行います。さらに、SRV\_MIGRATE\_RESUME イベント・ハンドラ内から結果をクライアントに引き続き送信する必要があるかどうかも判断する必要があります。

## エラー・メッセージ

接続マイグレーション機能を使用しているときに発生する可能性があるエラー・メッセージを次に示します。

エラー	説明
srv_thread_props(): プロパティ (SRV_T_SESSIONID) は利用できません。	クライアントがまだ取得していないセッション ID を取得しようとしました。
srv_send_ctlinfo(SRV_CTL_MIGRATE): 接続はマイグレートできません。	クライアントがマイグレーションをサポートしません。
srv_send_ctlinfo(SRV_CTL_MIGRATE): この時点でマイグレーションをキャンセルすることはできません。	すでに開始されたマイグレーションのキャンセルを要求しました。
マイグレーションが失敗しました。SRV_MIGRATE_STATE ハンドラはインストールされていません。	デフォルトの SRV_MIGRATE_STATE ハンドラがマイグレーション失敗を検出しました。

## CS\_BROWSEDESC 構造体

srv\_tabname と srv\_tabcolname は、CS\_BROWSEDESC 構造体を使ってブラウズ・モード・クエリの基本構造体に関する情報を返します。

CS\_BROWSEDESC 構造体の定義は次のとおりです。

```

/*
** CS_BROWSEDESC
** The Open Server browse column description
** structure.
*/
typedef struct _cs_browstedesc
{
    CS_INT status;
    CS_BOOL isbrowse;
    CS_CHAR origname[CS_MAX_NAME];
}

```

```

        CS_INT      orignlen;
        CS_INT      tablenum;
        CS_CHAR     tablename[CS_OBJ_NAME];
        CS_INT      tabnlen;
    } CS_BROWSEDESC;

```

各パラメータの意味は次のとおりです。

- **status** は、次に示す記号の OR (論理和) 演算の結果のビットマスクです。

**CS\_EXPRESSION** は、カラムが式 (次のクエリの "sum\*2" など) の結果であることを示します。

```
select sum*2 from areas
```

**CS\_RENAMED** は、カラムの見出しがカラムのオリジナル名ではないことを示します。カラムが次の形式のクエリの結果である場合には、データベース内のカラム名と異なった見出しになります。

```
select Author = au_lname from authors
```

- **isbrowse** は、ブラウザ・モードでカラムを更新できるかどうかを示します。

カラムが式の結果でもなく **timestamp** カラムでもない場合や、カラムがブラウザ可能なテーブルに所属している場合は、更新が可能です。ユニークなインデックスと **timestamp** カラムがあるテーブルは、ブラウザ可能です。

**isbrowse** は、カラムを更新できる場合は **CS\_TRUE** に、更新できない場合は **CS\_FALSE** に設定されます。

- **origname** は、データベース内のカラムのオリジナル名です。

カラムを更新する場合は、**select** 文で指定された見出しではなく、オリジナル名を使用する必要があります。

- **orignlen** は、**origname** の長さのバイト数です。
- **tablenum** は、カラムの所属しているテーブルの番号です。**select** 文の "from" リストの最初のテーブルがテーブル 1、次がテーブル 2 というようになります。
- **tablename** は、カラムが所属しているテーブルの名前です。
- **tabnlen** は、**tablename** の長さ (バイト数) です。



## CS\_DATAFMT 構造体

CS\_DATAFMT 構造体は、データ値とプログラム変数を記述するために使用します。次に例を示します。

- `srv_bind` は、CS\_DATAFMT 構造体を使って送信元または送信先のプログラム変数を記述します。
- `srv_descfmt` は、CS\_DATAFMT 構造体を使ってクライアント・データを記述します。
- `cs_convert` を実行するには、変換前と変換後のデータを記述する CS\_DATAFMT 構造体が必要です。

ほとんどのルーチンは、CS\_DATAFMT のフィールドのサブセットしか使用しません。たとえば、`srv_bind` は `name` および `usertype` フィールドを使用しません。また、`srv_descfmt` は `format` フィールドを使用しません。ルーチンが使用する CS\_DATAFMT のフィールドについては、そのルーチンのリファレンス・ページを参照してください。

CS\_DATAFMT 構造体の定義は次のとおりです。

```
typedef struct _cs_datafmt
{
    CS_CHAR      name[CS_MAX_NAME];    /* Name of data.*/
    CS_INT       namelen;              /* Length of name.*/
    CS_INT       datatype;             /* Datatype of data.*/
    CS_INT       format;               /* Format symbols.*/
    CS_INT       maxlength;            /* Max length of data.*/
    CS_INT       scale;                /* Scale of data.*/
    CS_INT       precision;            /* Precision of data.*/
    CS_INT       status;               /* Status symbols.*/

    /*
     ** The following field is not used in Open Server.
     ** It must be set to 1 or 0.
     */
    CS_INT       count;

    /*
     ** These fields are used to support user-defined
     ** datatypes and international datatypes:
     */
    CS_INT       usertype;              /* User-defined type.*/
    CS_LOCALE    *locale;              /* Locale information.*/
} CS_DATAFMT;
```

各パラメータの意味は次のとおりです。

- `name` はデータ名、つまりカラムまたはパラメータの名前です。
- `namelen` は `name` の長さ (バイト数) です。 `namelen` を `CS_NULLTERM` に設定し、`NULL` で終了する名前を示してください。 `name` が `NULL` の場合は、 `namelen` を 0 に設定します。
- `datatype` はデータのデータ型です。「[データ型](#)」(229 ページ) にリストされている `Open Server` データ型のいずれかです。

---

**注意** `datatype` フィールドには、データの `Open Server` データ型を記述します。 `usertype` は、`Open Server` データ型のほかにアプリケーション定義データ型がデータに設定されている場合にのみ使用します。

---

たとえば、次の `Adaptive Server Enterprise` コマンドは `Adaptive Server Enterprise` のユーザ定義タイプ `birthday` を作成します。

```
sp_addtype birthday, datatype
```

さらに、次のコマンドによって、この新しい型のカラムを含むテーブルが作成されます。

```
create table birthdays
(
    name          varchar(30),
    happyday      birthday
)
```

ユーザ定義のデータ型をサポートする `Open Server` アプリケーションは、`CS_DATAFMT` の `datatype` フィールドを `CS_DATETIME_TYPE` に、`usertype` フィールドを型 `birthday` のユーザ定義の ID に設定することによって、クライアントにこの情報を返します。

- `format` は、文字またはバイナリ・データの送信先のフォーマットを記述します。 `format` は、次の記号の `OR` (論理和) 演算の結果のビットマスクです。 [表 2-7](#) に、 `format` の有効値を示します。

表 2-7 : format の値 (CS\_DATAFMT)

記号	意味	注意
CS_FMT_NULLTERM	データは NULL で終了しなければならない。	文字または text データ
CS_FMT_PADBLANK	送信先変数の長さいっぱいまで、データの後にブランクを埋め込む必要がある。	文字または text データ
CS_FMT_PADNULL	送信先変数の長さいっぱいまで、データの後に NULL を埋め込む必要がある。	バイナリ、 <b>image</b> 、文字、または text データ
CS_FMT_UNUSED	このデータ型には埋め込みも NULL 終了も適用されない。	すべてのデータ型

- `maxlength` は、`CS_DATAFMT` を使用している Open Server ルーチンに応じて、さまざまな長さを表すことができます。表 2-8 に、`maxlength` が表す長さを示します。

表 2-8 : maxlength の意味 (CS\_DATAFMT)

Open Server ルーチン	maxlength の意味
<code>srv_bind</code>	バインド変数の長さ
<code>srv_descfmt</code>	記述されているカラムまたはパラメータの可能な最大長
<code>cs_convert</code>	変換元データの長さの変換先バッファ容量の長さ

- `scale` はデータの位取りです。データ型が `decimal` または `numeric` の場合だけ、`scale` を使用します。

`scale` の有効値は、`CS_MIN_SCALE` から `CS_MAX_SCALE` の範囲です。デフォルトの位取りは `CS_DEF_SCALE` です。

送信先データが送信元データと同じ位取りを使用することを示すには、`scale` を `CS_SRC_VALUE` に設定します。

- `scale` は、`precision` 以下でなければなりません。
- `precision` はデータの精度です。データ型が `decimal` または `numeric` の場合だけ、`precision` を使用します。

`precision` の有効値は、`CS_MIN_PREC` から `CS_MAX_PREC` の範囲です。デフォルトの精度は `CS_DEF_PREC` です。

送信先データが送信元データと同じ精度を使用することを示すには、`precision` を `CS_SRC_VALUE` に設定します。

- precision は、scale 以上でなければならない。
- status は、情報の種類を指定するために使用されるビットマスクです。表 2-9 に、status に指定できる情報の種類を示します。

表 2-9 : status の値 (CS\_DATAFMT)

記号値	意味
CS_CANBENULL	カラムは、NULL 値を含むことができる。
CS_DESCIN	CS_DATAFMT 構造体は、動的 SQL 入力パラメータを記述している。
CS_DESCOUT	CS_DATAFMT 構造体は、動的 SQL 出力パラメータを記述している。
CS_HIDDEN	カラムは公開された「隠しカラム」。
CS_INPUTVALUE	パラメータは、カーソル・オープン・コマンドまたは戻り値のない RPC パラメータのための入力値。
CS_KEY	カラムは、キー・カラムである。
CS_RETURN	パラメータは、RPC コマンドへの戻りパラメータ。
CS_TIMESTAMP	カラムは、 <i>timestamp</i> カラム。アプリケーションは、ブラウザ・モードで更新を行うときに <i>timestamp</i> カラムを使用する。
CS_UPDATABLE	カラムは、更新可能なカーソル・カラムである。
CS_UPDATECOL	パラメータは、カーソル宣言コマンドの <code>update</code> 句内のカラム名。
CS_VERSION_KEY	カラムは、ローのバージョン・キーの一部である。  Adaptive Server Enterprise は、バージョン・キーを位置付けのために使用する。
CS_NODEFAULT	パラメータに対して指定されたデフォルトはない。

- count は、Server Library ルーチンでは使用しません。常に 0 または 1 に設定してください。
- usertype は、返されるデータが存在する場合のユーザ定義のデータ型です。
- locale は、ローカライゼーション情報が格納されている CS\_LOCALE 構造体へのポインタです。ローカライゼーション情報が必要でない場合には、locale を NULL に設定してください。

## CS\_IODESC 構造体

CS\_IODESC は「I/O 記述子構造体」とも呼ばれ、text または image データを記述します。

クライアントの text または image データを処理するときには、Open Server アプリケーションは cmd 引数を CS\_GET に設定して srv\_text\_info を呼び出します。この呼び出しでデータが挿入されるのは、CS\_IODESC 引数の total\_txtlen フィールドだけです。

クライアントにデータ・カラムを送信する場合には、アプリケーションは、cmd 引数を CS\_SET に設定して srv\_text\_info を呼び出します。この場合、CS\_IODESC 構造体は、送信されている text または image カラムを記述しています。CS\_IODESC は下記のように定義されています。

```
typedef struct _cs_iodesc
{
    CS_INT    iotype;                /* CS_IODATA          */
    CS_INT    datatype;             /* Text or image.*/
    CS_LOCALE *locale;              /* Locale information.*/
    CS_INT    usertype;             /* User-defined type.*/
    CS_INT    total_txtlen;         /* Total data length.*/
    CS_INT    offset;               /* Reserved.*/
    CS_BOOL   log_on_update;        /* Log the insert.*/
    CS_CHAR   name[CS_OBJ_NAME];    /* Name of data object.*/
    CS_INT    namelen;              /* Length of name.*/
    CS_BYTE   timestamp[CS_TS_SIZE]; /* Adaptive Server Enterprise id. */
    CS_INT    timestamplen;         /* Length of timestamp.*/
    CS_BYTE   textptr[CS_TP_SIZE];  /* Adaptive Server Enterprise pt */
    CS_INT    textptrlen;           /* Length of textptr.*/
} CS_IODESC;
```

各パラメータの意味は次のとおりです。

- **iotype** は実行する I/O のタイプを指定します。text および image のオペレーションでは、iotype には常に値 CS\_IODATA が設定されます。
- **datatype** は、データ・オブジェクトの「**データ型**」です。datatype の有効値は、CS\_TEXT\_TYPE と CS\_IMAGE\_TYPE だけです。
- **locale** は、現在、Open Server では使用されません。NULL に設定します。
- **usertype** は、Open Server では使用されません。
- **total\_txtlen** は、text または image 値の全体の長さ (バイト数) です。
- **offset** は、今後の使用のために予約されています。

- `log_on_update` は、この `text` または `image` 値に対する更新をログに記録するかどうかを表します。
- `name` は、`text` または `image` カラムの名前です。
- `namelen` は、`name` の長さ (バイト数) です。 `CS_NULLTERM` の場合は、名前が `null` で終了することを示します。
- `timestamp` は、カラムのテキスト・タイムスタンプです。テキスト・タイムスタンプは、`text` または `image` カラムが最後に修正された時間を示します。
- `timestamplen` は、`timestamp` の長さ (バイト数) です。
- `textptr` は、カラムの挿入または取得のための `text` または `image` バイトの配列です。
- `textptrlen` は、`textptr` の長さ (バイト数) です。

## CS-Library

**CS-Library** は、Open Server アプリケーションと Open Client アプリケーションの両方で有用あるいは必要であるユーティリティ・ルーチンおよび構造体の集まりです。以前のバージョンでは、**Server-Library** と **Client-Library** の両方でこのようなユーティリティ・ルーチンと構造体を提供していたため、無駄な重複がありました。

### 共通ルーチン

CS-Library のルーチンは、次の機能をサポートします。

- データ型変換
- 算術演算
- 文字セット変換
- 日時オペレーション
- ソート順オペレーション
- ローライゼーション・ルーチン

CS-Library には、CS-Library の構造体を割り付けるルーチンも含まれています。

スタンドアロンの CS-Library アプリケーションを作成することもできますが、このライブラリの主な機能は、Open Client アプリケーションと Open Server アプリケーションに共通のユーティリティを提供することです。

これらのルーチンが提供する機能には、現行の Server-Library ルーチンが提供しているものもあります。Server-Library ルーチンに対応する CS-Library ルーチンに置き換える必要は現時点ではありませんが、将来的には必要になる可能性があります。

## 共通データ構造体

共通ルーチンに加えて、CS-Library は Open Client と Open Server の両方のアプリケーションに役立つデータ構造体を提供しています。これらのデータ構造体の中には、アプリケーション・プログラミング環境、つまり「コンテキスト」に関する情報が格納された CS\_CONTEXT 構造体が含まれています。

Open Server アプリケーションのプログラマは、この構造体に格納されているグローバル・アプリケーション属性を設定することで、アプリケーションの動作を調整できます。この機能の詳細については、「[プロパティ](#)」(157 ページ)を参照してください。

他の CS-Library 構造体は、Open Client とアプリケーションと Open Server アプリケーションの間で通信されるデータについての情報を含んでいます。

---

**注意** Client-Library プログラムと Server-Library プログラムは、コンテキスト構造体を必要とし、CS-Library だけがこの構造体を割り付けることができます。そのため、Client-Library プログラムと Server-Library プログラムはすべて、CS-Library への呼び出しを少なくとも 2 つ、つまり、CS\_CONTEXT を割り付ける呼び出しとそれを解除する呼び出しを含んでいます。

---

## エラー処理

Open Server アプリケーションは、CS-Library エラーを報告するために、`cs_config` ルーチンで「メッセージ・コールバック・ルーチン」をインストールしなければなりません。`srv_props` でインストールした標準の Open Server エラー・ハンドラは、`cs_convert` への呼び出しで発生したデータ変換エラーなどの CS-Library エラーを捕捉しません。

Open Server アプリケーションが CS-Library ハンドラをインストールしていない場合、Open Server は、アプリケーションが `srv_version` を呼び出したときにデフォルト・ハンドラをインストールします。このデフォルト・ハンドラは、Open Server のログに CS-Library エラーを書き込みます。

CS-Library エラーの処理の詳細と CS-Library の全般的な情報については、『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

## CS\_SERVERMSG 構造体

CS\_SERVERMSG 構造体は、サーバ・エラー・メッセージに関する情報を含んでいます。

Open Server では、CS\_SERVERMSG 構造体を使用し、`srv_sendinfo` ルーチンを介してクライアントにエラー・メッセージを送信します。

CS\_SERVERMSG 構造体の定義は次のとおりです。

```
/*
** CS_SERVERMSG
** The server message structure.
*/

typedef struct _cs_servermsg
{
    CS_INT    msgnumber;
    CS_INT    state;
    CS_INT    severity;
    CS_CHAR   text[CS_MAX_MSG];
    CS_INT    textlen;
    CS_CHAR   svrname[CS_MAX_NAME];
    CS_INT    svrlen;

    /*
    ** If the error involved a stored procedure,
```



```

    ** the following fields contain information
    ** about the procedure:
    */
    CS_CHAR  proc[CS_MAX_NAME];
    CS_INT   proclen;
    CS_INT   line;

    /*
    ** Other information.
    */
    CS_INT   status;
    CS_BYTE  sqlstate[CS_SQLSTATE_SIZE];
    CS_INT   sqlstatelen;
} CS_SERVERMSG;

```

各パラメータの意味は次のとおりです。

- `msgnumber` は、クライアントに報告される Open Server またはアプリケーションの「メッセージ番号」です。
- `state` は、メッセージ生成時の状態です。アプリケーションによって定義します。
- `severity` は、メッセージの重大度です。
- `text` は、メッセージ・テキストです。
- `textlen` は、`text` のバイト単位での長さです。
- `svrname` は、メッセージを生成したサーバ名です。この値を、現在実行中の Open Server アプリケーションの名前などの別の名前にすることもできます。
- `svrlen` は `svrname` の長さ (バイト数) です。
- `proc` は、メッセージを生成した「ストアド・プロシージャ」の名前です (存在する場合)。
- `proclen` は、`proc` の長さ (バイト数) です。
- `line` は、メッセージを生成したストアド・プロシージャ内の行番号です (存在する場合)。
- `status` は、メッセージのまとまりがメッセージの最初、最後、中間のどの部分であるか、メッセージに拡張エラー・データが含まれているかどうかについての情報を含んでいます。`status` は、バイト順フラグなので、2つ以上の値を設定できます。次に例を示します。

```
mrec.status = CS_FIRST_CHUNK | CS_LAST_CHUNK;
```

mrec は、CS\_SERVERMSG 構造体として宣言されます。

表 2-10 に、status の有効値を示します。

**表 2-10 : CS\_SERVERMSG 構造体の status フィールドの値**

値	意味
CS_HASEED	メッセージに関連する拡張エラー・データがある。
CS_FIRST_CHUNK	text に含まれるメッセージ・テキストは、メッセージの最初のまとまり。  CS_FIRST_CHUNK と CS_LAST_CHUNK が両方ともオンになっていると、text にはメッセージ全体が含まれる。  CS_FIRST_CHUNK も CS_LAST_CHUNK もオンになっていないと、text にはメッセージの中間のまとまりが含まれる。
CS_LAST_CHUNK	text に含まれるメッセージ・テキストは、メッセージの最後のまとまり。  CS_FIRST_CHUNK と CS_LAST_CHUNK が両方ともオンになっていると、text にはメッセージ全体が含まれる。  CS_FIRST_CHUNK も CS_LAST_CHUNK もオンになっていないと、text にはメッセージの中間のまとまりが含まれる。

- sqlstate は、エラーを説明するバイト文字列です。  
すべてのサーバ・メッセージが SQL ステータス値を持っているわけではありません。メッセージに SQL ステータス値がない場合、sqlstate の値は "ZZZZZ" になります。
- sqlstatelen は、sqlstate 文字列の長さ (バイト数) です。  
「クライアント・コマンド・エラー」(38 ページ) を参照してください。

## カーソル

Adaptive Server Enterprise が実装しているカーソルは Server-Library と Client-Library によってサポートされています。

Adaptive Server Enterprise にカーソルがどのように実装されているかについては、『ASE リファレンス・マニュアル』を参照してください。

Client-Library でカーソルがどのようにサポートされているかについては、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

## カーソルの概要

カーソルとは、SQL 文にリンクされた記号名です。カーソルを宣言すると、リンクが確立されます。SQL 文は、次のいずれかです。

- SQL select 文
- Transact-SQL execute 文
- 動的 SQL 準備文

カーソルに関連付けられた SQL 文は、カーソルの本体と呼ばれます。クライアントがカーソルをオープンすると、カーソルの本体が実行され、結果セットが生成されます。Open Server アプリケーションは、カーソル要求を検出してカーソル結果をクライアントに返します。

## カーソルの利点

カーソルを使うことによって、クライアント・アプリケーションは、単にデータ・ローの完全なセットを取得するだけでなく、結果セット内の個々のローにアクセスできます。

1つの接続で同時に複数のカーソルをオープンすることもできます。すべてのカーソル結果セットを同時にアプリケーションで使用できるため、アプリケーションは、随時、カーソル結果セットからデータ・ローをフェッチすることができます。これは、1ローずつ順番に処理しなければならない他のタイプの結果セットとは対照的な方法です。

また、クライアント・アプリケーションは、カーソル結果セットのローをフェッチしながら基本データベース・テーブルを更新することができます。

## Open Server アプリケーションとカーソル

この項では、Open Server のカーソル・サポートの基本的な情報について説明します。SRV\_CURSOR イベント・ハンドラの構築方法の詳細については、「[特定の要求への応答方法](#)」(77 ページ)を参照してください。

### カーソル要求の生成方法

クライアント・アプリケーションは、Open Server アプリケーションにカーソル・コマンドを発行してカーソルを要求します。

クライアント・アプリケーションは、Client-Library コマンドの `ct_cursor` を呼び出してカーソル・コマンドを開始します。『Open Client Library/C リファレンス・マニュアル』を参照してください。

カーソルを要求すると、Open Server は SRV\_CURSOR イベントを生成します。カーソル要求に応答する場合、Open Server アプリケーションには SRV\_CURSOR イベント・ハンドラが必要です。

### カーソル・コマンドのタイプ

表 2-11 に、クライアントが発行できるカーソル・コマンドのタイプを示します。

表 2-11 : カーソル・コマンドの概要

コマンドのタイプ	実行する内容
宣言	カーソル名をカーソルの本体と関連付ける。
オープン	カーソルの本体を実行し、カーソル結果セットを生成する。
情報	カーソルのステータスを報告するか、カーソル・ロー・フェッチ・カウントを設定する。
フェッチ	カーソル結果セットからローをフェッチする。
更新または削除	現在のカーソル・ローの内容を更新または削除する。
クローズ	カーソル結果セットを使用できないようにする。カーソルを再オープンすると、カーソル結果セットが再生成される。
割り付け解除	カーソルを存在しないようにする。割り付け解除されたカーソルを再オープンすることはできない。

通常、クライアント・アプリケーションは表 2-11 にリストされた順序でカーソル・コマンドを発行しますが、必要に応じてこの順序が変わることもあります。たとえば、カーソルをフェッチしてからクローズし、再びオープンしてもう一度ローをフェッチする、というような場合です。

## クライアントとのカーソル情報の交換

SRV\_CURSOR イベント・ハンドラは、`srv_cursor_props` ルーチンと `SRV_CURDESC` 構造体を使って、カーソル情報をクライアントと交換します。`srv_cursor_props` では、現在の情報をクライアントに送信してクライアントからカーソル情報を取得する場合、`SRV_CURDESC` 構造体にアクセスします。

「`srv_cursor_props`」(288 ページ) を参照してください。

クライアントとサーバは、1 つの接続で複数のカーソル情報を交換できるので、それぞれのカーソルを識別する必要があります。`Open Server` アプリケーションは、カーソル宣言への応答として、ユニークなカーソル ID を送り返します。このカーソルが存在する間は、クライアントとサーバはこの ID を使用してこのカーソルを参照します。

## SRV\_CURDESC 構造体

SRV\_CURDESC 構造体には、次のようなカーソル情報が含まれます。

- ユニークなカーソル ID

- クライアントが最後に発行したカーソル・コマンドのタイプ
- カーソルのステータス

SRV\_CURDESC 構造体の定義は次のとおりです。

```

/*
** SRV_CURDESC
** The Open Server cursor description
** structure.
*/

typedef struct srv_curdesc
{
    CS_INT        curid;
    CS_INT        numupcols;
    CS_INT        fetchcnt;
    CS_INT        curstatus;
    CS_INT        curcmd;
    CS_INT        cmdoptions;
    CS_INT        fetchtype;
    CS_INT        rowoffset;
    CS_INT        curnamelen;
    CS_CHAR       curname[CS_MAX_CHAR];
    CS_INT        tabnamelen;
    CS_CHAR       tabname[CS_MAX_CHAR];
    CS_VOID       *userdata;

} SRV_CURDESC;

```

表 2-12 に、SRV\_CURDESC 構造体のフィールドのリストを示します。

表 2-12 : SRV\_CURDESC 構造体のフィールド

フィールド名	説明	注意
curid	現在のカーソル識別子	Open Server アプリケーションは、クライアントからの CS_CURSOR_DECLARE コマンドに 応答するとき curid を設定しなければならない。その後クライアントから発行される、宣言されたカーソルに関連するコマンドはすべて、識別子として curid を使用しなければならない。現在のカーソル識別子が ないか、クライアントがすべての使用可能なカーソルのステータスを要求している場合には、curid は 0 に設定される。
numupcols	カーソル更新句内の カラム数	numupcols は、更新カラムがない場合には 0 に設定される。この情報は、カーソルが宣言されるときに使用できる。
fetchcnt	このカーソルに関する現在のロー・ フェッチ・カウン ト、つまり CS_CURSOR_FETCH コマンドに 応答してクライアントに 送信されるローの 数	fetchcnt は、CS_CURSOR_INFO コマンドをクライアントから受け取るか、このコマンドに 応答するクライアントに CS_CURSOR_INFO コマンドを送信するとき、設定される。クライアントでロー・フェッチ・カウン トが明示的に設定されていない場合は、fetchcnt は 1 に設定される。Open Server アプリケーションが、要求されたフェッチ・ カウントをサポートできない場合は、応答する前にこのフィールドを別の値に設定することができる。
curstatus	現在のカーソルの ステータス	Open Server は、クライアントから受け取ったカーソル・ コマンドに 応答してカーソル・ ステータスを設定する。有効な値のリストについては、 「curstatus の値」(73 ページ)を参照。
curcmd	現在のカーソル・ コマンド・ タイプ	有効な値のリストについては、 表 2-14 を参照。
cmdoptions	カーソル・ コマンドに関する オプション	すべてのコマンドに関連オプションがあるわけではない。cmdoptions の値は、カーソル・ コマンドのタイプによって異なる。コマンド別の cmdoptions の値のリストについては、 表 2-14 を参照。

フィールド名	説明	注意
fetchtype	クライアントが要求したフェッチのタイプ	<p>fetchtype は、クライアントから CS_CURSOR_FETCH コマンドを受け取ったときに書き込まれる。有効なフェッチ・タイプとその意味は次のとおり。</p> <ul style="list-style-type: none"> <li>• CS_NEXT – 次のロー</li> <li>• CS_PREV – 前のロー</li> <li>• CS_FIRST – 最初のロー</li> <li>• CS_LAST – 最後のロー</li> <li>• CS_ABSOLUTE – rowoffset フィールドで識別されるロー</li> <li>• CS_RELATIVE – 現在のローに対して rowoffset フィールドの値を加減算したもの</li> </ul> <p>Adaptive Server Enterprise への要求は必ず CS_NEXT の fetchtype である。</p>
rowoffset	CS_ABSOLUTE または CS_RELATIVE フェッチに関するロー位置	<p>rowoffset は、他のすべてのフェッチ・タイプに対して定義されていない。rowoffset は、クライアントから CS_CURSOR_FETCH コマンドを受け取ったときに設定される。</p>
curnamelen	curname 内のカーソル名の長さ	<p>curname が有効でない場合、curnamelen はゼロになります。curnamelen は、カーソル名の長さを返します。</p>
curname	現在のカーソルの名前	
tabnamelen	tablename のテーブル名の長さ	<p>tablename が有効でない場合、tabnamelen はゼロになります。tabnamelen はテーブル名の長さを返します。tabnamelen は、CS_CURSOR_UPDATE コマンドまたは CS_CURSOR_DELETE コマンドがクライアントから受信されたときに記述されます。</p>
tablename	カーソルの update または delete コマンドに関連するテーブル名	<p>tablename は、カーソルの update コマンドまたは delete コマンドに関連付けられているテーブル名。tablename は、クライアントから CS_CURSOR_UPDATE コマンドまたは CS_CURSOR_DELETE コマンドを受け取ったときに書き込まれる。</p>



フィールド名	説明	注意
userdata	ユーザ用のデータ領域に対するポインタ	このフィールドを使用すると、グローバル変数や静的な変数を使用しなくてもデータを特定のカーソルに関連付けることができる。Open Server は、userdata を操作しない。このコマンドは、Open Server アプリケーション・プログラマのために提供されている。

### **curstatus の値**

SRV\_CURDESC 構造体の **curstatus** フィールドは、次の表に示すどのような値の組み合わせでも可能なビットマスクです。

表 2-13 : curstatus の値 (SRV\_CURDESC)

値	意味
CS_CURSTAT_DECLARED	カーソルが宣言された。このステータスは、次のカーソル・コマンドが処理された後にリセットされる。
CS_CURSTAT_OPEN	カーソルがオープンされた。
CS_CURSTAT_ROWCONT	カーソルが、CS_CURSOR_FETCH コマンドで返されるローの数を指定した。
CS_CURSTAT_RDONLY	カーソルは、読み込み専用なので更新できない。CS_CURSOR_UPDATE または CS_CURSOR_DELETE を受信した場合は、Open Server アプリケーションからクライアントにこのカーソルについてのエラーが返される。
CS_CURSTAT_UPDATABLE	カーソルを更新できる。
CS_CURSTAT_CLOSED	カーソルはクローズされたが、割り付け解除されていない。後で再オープンできる。カーソル宣言においても、このステータスが設定される。Open Server は、CS_CURSOR_OPEN を受信した場合はこの値をクリアし、CS_CURSOR_CLOSE を受信した場合はこの値を再設定する。
CS_CURSTAT_DEALLOC	カーソルがクローズされ、割り付け解除された。このとき、他のステータス・フラグは設定されない。

### curcmd の値

SRV\_CURDESC 構造体の curcmd フィールドには、表 2-14 に示す値を設定できます。この表には、関連する cmdoptions の値も示します。

表 2-14 : curcmd の値 (SRV\_CURDESC)

値	意味	cmdoptions の有効値
CS_CURSOR_CLOSE	カーソルの close コマンド。	SRV_CUR_DEALLOC または SRV_CUR_UNUSED。 SRV_CUR_DEALLOC は、カーソルが再オープンされないことを示している。 Open Server アプリケーションは、関連するカーソル・リソースをすべて削除しなければならない。 カーソル ID 番号は再使用できる。
CS_CURSOR_DECLARE	カーソルの declare コマンド。アプリケーションは、 <code>srv_langlen</code> と <code>srv_langcpy</code> を介してカーソル文の実際のテキストを取得できる。	SRV_CUR_UPDATABLE、SRV_CUR_RDONLY、または SRV_CUR_DYNAMIC。 SRV_CUR_DYNAMIC は、動的に準備された SQL 文に対してクライアントがカーソルを宣言することを示す。この場合、カーソル文のテキストは、実際には、作成された文の名前になる。
CS_CURSOR_DELETE	カーソルの delete コマンド。カーソルを使用して定位置ローの delete を実行する。	このコマンドには有効なオプションはない。 cmdoptions の値は常に SRV_CUR_UNUSED。
CS_CURSOR_FETCH	カーソルの fetch コマンド。カーソルを使用してローの fetch を実行する。	このコマンドには有効なオプションはない。 cmdoptions の値は常に SRV_CUR_UNUSED。

値	意味	cmdoptions の有効値
CS_CURSOR_INFO	<p>カーソル情報コマンド。クライアントは、このコマンドを Open Server アプリケーションに送信し、カーソル・ロー・フェッチ・カウントを設定するか、カーソル・ステータス情報を要求する。Open Server アプリケーションは、現在のカーソルを記述するために、任意のカーソル・コマンド (CS_CURSOR_INFO 自体も含む) に応答して、クライアントにこのコマンドを送信する。</p>	<p>クライアントが現在のロー・フェッチ・カウントを書き込む場合は、SRV_CUR_SETROWS。fetchcnt フィールドには、要求されたフェッチ・カウントが含まれている。クライアントが現在のカーソルのステータス情報を要求する場合は、SRV_CUR_ASKSTATUS。クライアントがアテンション送信後に使用可能なカーソルを調べたいときに、この値が発生する。curid フィールドには 0 が含まれる。Open Server アプリケーションは、現在使用可能な各カーソルに対して CS_CURSOR_INFO 応答を戻す必要がある。</p> <p>Open Server アプリケーションが CS_CURSOR_INFO コマンドに応答する場合は、SRV_CUR_INFORMSTATUS。curstatus フィールドには、カーソル・ステータスが含まれる。</p>
CS_CURSOR_OPEN	<p>カーソルの open コマンド。</p>	<p>SRV_CUR_HASARGS または SRV_CUR_UNUSED。</p>

値	意味	cmdoptions の有効値
CS_CURSOR_UPDATE	カーソルの update コマンド。カーソルを使用して定位ローの update を実行する。Open Server アプリケーションは、 <code>srv_langlen</code> と <code>srv_langcpy</code> を呼び出して、カーソルの update 文の実際のテキストを取得できる。	SRV_CUR_HASARGS または SRV_CUR_UNUSED。

## カーソル要求の処理

Open Server アプリケーションは、SRV\_CURSOR イベント・ハンドラを使用してカーソル要求を処理します。このハンドラには、発行されたカーソル・コマンドを検出したり、適切な情報とともに応答したりするコードが含まれています。

最初に、イベント・ハンドラは `cmd` 引数を `CS_GET` に設定して `srv_cursor_props` を呼び出し、現在のカーソルと、SRV\_CURSOR イベントをトリガしたカーソル・コマンドを特定します。次に、Open Server は、Open Server アプリケーションの SRV\_CURDESC 構造体の `curcmd` フィールドにコマンド・タイプを格納します。

この時点で他に取得する必要がある情報やクライアントに返信するデータがあれば、アプリケーションはそれを決定できます。場合によっては、パラメータ・フォーマットとパラメータを取得しなければならないことがあります。また、必要ならば、現在のカーソルのステータスとフェッチするロー数を取得しなければなりません。

CS\_CURSOR\_INFO コマンドを送り返すだけで済む場合もありますが、結果データまたは戻りパラメータを送信しなければならないこともあります。

## 特定の要求への応答方法

この項では、SRV\_CURSOR イベント・ハンドラが、特定のタイプのカーソル要求に応答する方法について説明します。

cmd を CS\_SET に設定して srv\_cursor\_props を呼び出す前に、Open Server アプリケーションは SRV\_CURDESC 構造体の curid フィールドおよびその他の関係するフィールドを設定しておく必要がある点に注意してください。

表 2-15 に、クライアントと Open Server アプリケーション間のカーソル要求と応答の有効な交換についてまとめてあります。右矢印 (→) は、cmd が CS\_GET に設定されている (Open Server アプリケーションはクライアントから情報を取得する) ことを示しています。左矢印 (←) は、cmd が CS\_SET に設定されている (Open Server アプリケーションは情報をクライアントに送信する) ことを示しています。

**表 2-15 : 有効なカーソル要求と応答**

クライアントのアクション	Open Server アプリケーションの応答
カーソルの宣言 (SRV_CURDESC の curcmd フィールドは CS_CURSOR_DECLARE を含んでいる)	→ SRV_CURDESC から curcmd 値を取得する。 (srv_cursor_props)  → カーソル・パラメータの数を取得する (存在する場合)。 (srv_numparams)  → カーソル・パラメータのフォーマットを取得する (存在する場合)。 (type 引数を SRV_CURDATA に設定した srv_descfmt)  → 更新カラム情報を取得する (存在する場合)。 (type 引数を SRV_UPCOLDATA に設定した srv_descfmt)  → カーソル・コマンドの実際のテキストを取得する。 (srv_langlen と srv_langcpy)  ← カーソル ID を設定する。curcmd フィールドを CS_CURSOR_INFO に、curid フィールドをユニークなカーソル ID に設定する。 (srv_cursor_props)  ← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)

クライアントのアクション	Open Server アプリケーションの応答
<p>現在のカーソルのステータスの要求または フェッチ・カウントの送信 (SRV_CURDESC の curcmd フィールドは CS_CURSOR_INFO を含んでいる)</p>	<p>→ SRV_CURDESC 構造体から curcmd、curid cmdoptions の値を取得する。 (srv_cursor_props)</p> <p>← クライアント側で cmdoptions フィールドが SRV_CUR_SETROWS に設定された場合、1 回の フェッチで返されるローの数を送信する。</p> <p>(curcmd を CS_CURSOR_INFO に設定した srv_cursor_props)</p> <p>← クライアント側で cmdoptions フィールドが SRV_CUR_ASKSTATUS に設定された場合、使用可 能なすべてのカーソルのステータスを送信する。 curcmd フィールドを CS_CURSOR_INFO に、curid フィールドをカーソル ID に設定する。</p> <p>(アクティブなカーソル、つまり、宣言、オープン、 またはクローズされたカーソルごとに srv_cursor_props を 1 回ずつ)</p> <p>← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>

クライアントのアクション	Open Server アプリケーションの応答
<p>カーソルのオープン (SRV_CURDESC の curcmd フィールドは CS_CURSOR_OPEN を含んでいる)</p>	<p>→ SRV_CURDESC 構造体から curcmd 値と curid 値を取得する。 (srv_cursor_props)</p> <p>→ カーソル・パラメータの数を取得する (存在する場合)。 (srv_numparams)</p> <p>→ カーソル・パラメータと実際のパラメータのフォーマットを取得する (存在する場合)。 (type 引数を SRV_CURDATA に設定した srv_descfmt、srv_bind、srv_xferdata)</p> <p>← カーソル・ステータスを送信する。curid を現在のカーソル ID に、curcmd を CS_CURSOR_INFO に設定する。 (srv_cursor_props)</p> <p>← 結果ロー・フォーマットを記述する。 (type 引数を SRV_ROWDATA に設定した srv_descfmt)</p> <p>← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>
<p>ローのフェッチ (SRV_CURDESC の curcmd フィールドは CS_CURSOR_FETCH を含んでいる)</p>	<p>→ SRV_CURDESC 構造体から curcmd 値と curid 値を取得する。 (srv_cursor_props)</p> <p>← 結果ローを fetchcnt の数だけ送信する。 (type 引数を SRV_ROWDATA に設定した srv_bind、srv_xferdata)</p> <p>← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>



クライアントのアクション	Open Server アプリケーションの応答
<p>カーソル更新コマンドの発行 (SRV_CURDESC の curcmd フィールドは CS_CURSOR_UPDATE を含んでいる)</p> <p>または</p> <p>カーソル削除コマンドの発行 (SRV_CURDESC の curcmd フィールドは CS_CURSOR_DELETE を含んでいる)</p>	<p>→ SRV_CURDESC 構造体から curcmd 値と curid 値を取得する。 (srv_cursor_props)</p> <p>→ 現在のローのキー・カラムを取得する。 (type 引数を SRV_KEYDATA に設定した srv_descfmt、srv_bind、srv_xferdata)</p> <p>→ curcmd が CS_CURSOR_UPDATE の場合は、更新値の数を取得する。 (srv_numparams)</p> <p>curcmd が CS_CURSOR_UPDATE の場合は、更新文の実際のテキストを取得する。 (srv_langlen と srv_langcpy)</p> <p>→ curcmd が CS_CURSOR_UPDATE の場合は、更新値を取得する。 (type 引数を SRV_CURDATA に設定した srv_descfmt、srv_bind、srv_xferdata )</p> <p>← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>
<p>カーソル・クローズ・コマンドの送信 (SRV_CURDESC の curcmd フィールドは CS_CURSOR_CLOSE を含んでいる)</p>	<p>→ SRV_CURDESC 構造体から curcmd 値と curid 値を取得する。 (srv_cursor_props)</p> <p>← カーソル・ステータスを送信する。 (srv_cursor_props)</p> <p>← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>

次のことに注意してください。

- Open Server アプリケーションがカーソル・コマンドに応答する場合、応答を終了するときには、必ず `status` 引数を `"SRV_DONE_FINAL"` に設定した `srv_senddone` を呼び出します。
- Open Server アプリケーションが、`cmd` を `"SET"` に設定した最初の `srv_cursor_props` コマンドを発行した後、`status` 引数を `SRV_DONE_FINAL` に設定して `srv_senddone` を発行するまでの間は、このアプリケーションから送信されるすべての情報がこのカーソルに適用されます。
- 内部的には、Open Server はクライアントがカーソルを宣言するときに受けるパラメータ・フォーマットを、クライアントがカーソルをオープンするときに受けるものと入れ替えます。この手順は、受けたパラメータのフォーマットが、パラメータ宣言のフォーマットと正確には同じでない場合に必要となります。たとえば、`CS_INT` としてパラメータを宣言しても、カーソルがオープンされるときに渡されるパラメータの型が `CS_SMALLINT` であることもあります。
- `srv_xferdata` は 1 つのデータ・ローを送信するため、`CS_CURSOR_FETCH` コマンドに応答するときは、現在のカーソルのロー・フェッチ・カウントの回数だけこのコマンドを呼び出します。

## キー・データ

キーは、ローをユニークに識別するロー・データのサブセットです。キー・データは、オープンされたカーソル内の「現在のロー」をユニークに記述します。これは、`CS_CURSOR_DELETE` または `CS_CURSOR_UPDATE` コマンドを処理するために使われます。カラムがキー・カラムである場合は、そのカラムを記述している `CS_DATAFMT` 構造体の `status` フィールドには `CS_KEY` ビットマスク・セットが設定されています。

## 更新カラム

クライアントがカーソルを「更新用」と宣言した場合は、`SRV_CURDESC` 構造体の `cmdoptions` フィールドは `CS_FOR_UPDATE` に設定され、`numupcols` フィールドはそのカーソルと関連する更新カラムの数に設定されます。

## 例

サンプル *ctos.c* には、カーソル・コマンド処理のコードが記述されています。

## スクロール可能カーソル

スクロール可能カーソルの機能を利用すると、FETCH 文で NEXT 句、PREVIOUS 句、FIRST 句、LAST 句、ABSOLUTE 句、または RELATIVE 句を指定することで、結果セットの任意の場所に現在位置を設定することができます。実装されるスクロール可能カーソルは読み込み専用で、INSENSITIVE プロパティまたは SEMI\_SENSITIVE プロパティが設定されます。

非スクロール可能で非反映型のカーソルも Open Server でサポートされており、CS\_NOSCROLL\_INSENSITIVE オプションを使用して設定されます。

Open Server に接続するリモート・クライアントは、ログイン時に、CS\_REQ\_CURINFO3 を使用してスクロール可能カーソルのサポートを要求できます。

## SRV\_CURDESC2 構造体

Open Server の SRV\_CURDESC2 スクロール可能カーソル構造体は、「[SRV\\_CURDESC 構造体](#)」(69 ページ) で説明している SRV\_CURDESC カーソル構造体のスーパーセットです。

[表 2-12](#) に示すフィールドの他に、[表 2-16](#) に示すフィールドが SRV\_CURDESC2 構造体に追加されています。

表 2-16 : SRV\_CURDESC2 構造体の追加フィールド

フィールド名	説明
currow_pos	カーソルの現在のロー位置。
curtotalrowcount	結果セット内のローの総数 (非反映型のスクロール可能カーソルの場合のみ)。

## curstatus の値

表 2-13 に示したオプションに加えて、次に示すカーソル宣言関連のオプションが SRV\_CURDESC2 の *curstatus* フィールドで使用可能です。

表 2-17 : curstatus の値 (SRV\_CURDESC2)

値	意味
CS_CURSTAT_SCROLLABLE	読み込み専用の、非反映型スクロール可能カーソル。
CS_CURSTAT_INSENSITIVE	読み込み専用の、非スクロール可能、非反映型カーソル。 このカーソルを指定する場合は、CS_CURSTAT_INSENSITIVE を有効にし、CS_CURSTAT_SCROLLABLE を無効にする必要があります。 非反映型のスクロール可能カーソルを指定する場合は、CS_CURSTAT_INSENSITIVE と CS_CURSTAT_SCROLLABLE の両方を有効にする必要があります。
CS_CURSTAT_SEMISENSITIVE	読み込み専用の、半反映型スクロール可能カーソル。 このカーソルを指定する場合は、CS_CURSTAT_SCROLLABLE も有効にする必要があります。
CS_CUR_RELOCKS_ONCLOSE	カーソルは終了し、共有読み込みロックが解放されます。

## curcmd の値

表 2-14 に示した値に加えて、表 2-18 に示す値が SRV\_CURDESC2 構造体の *curcmd* フィールドで使用可能です。この表には、関連する *cmdoptions* の値も示します。

表 2-18 : curcmd の値 (SRV\_CURDESC2)

値	意味	cmdoptions の有効値
CS_NOSCROLL_ INSENSITIVE	非スクロール可能、 非反映型のカーソ ル。	このコマンドには有効な オプションはない。 cmdoptions の値は常に SRV_CUR_UNUSED。  <b>注意</b> CTOS アプリケー ションを使用する場合、 非スクロール可能カーソ ルでは ct_scroll_fetch ルー チンを使用しないでくだ さい。代わりに、ct_fetch ルーチンを使用してくだ さい。
CS_CURSOR_ DECLARE	スクロール可能 カーソルのコマン ド・オプション。	SRV_CUR_SCROLL, SRV_CUR_SCROLL_ INSENS, SRV_CUR_ SCROLL_SEMISENS, SRV_CUR_NOSCROLL_ INSENS.  これらの cmdoptions は カーソルの declare サイク ルにおいてのみ有効であ り、SRV_CURDESC2 構 造体の curcmd フィールド には、ct_cursor を発行し ているリモート・クライ アントに基づいて、これ らのオプションのいずれ かが格納されます。

### srv\_cursor\_props2 ルーチン

SRV\_CURDESC2 構造体をサポートするために、srv\_cursor\_props2 ルーチンが Open Server に追加されています。

15.0 より前のアプリケーションで、CS\_VERSION\_125 を設定する場合は、SRV\_CURDESC 構造体と srv\_cursor\_props ルーチンを使用する必要があります。

Open Server でスクロール可能カーソルをサポートするバージョン 15.0 以降のアプリケーションでは、SRV\_CURDESC2 構造体を使用します。また、アプリケーションを CS\_VERSION\_xxx に設定します。ここで、xxx は Open Server のバージョンに対応します。

srv\_cursor\_props2 の引数は次のとおりです。

```
ret = srv_cursor_props2(SRV_PROC *spp, CS_INT cmd,  
SRV_CURDESC2 *cdp);
```

## データ・ストリーム・メッセージ

### データ・ストリーム・メッセージとは

データ・ストリーム・メッセージは、クライアントと Open Server アプリケーションとが情報を交換するための方法を提供します。

RPC も、機能は同様ですが、クライアントからサーバへの一方のみです。メッセージはさまざまな通信の用途に適合させることによって、両方向で機能します。たとえば、Sybase では、ログイン時にセキュリティ・ハンドシェイクを実行するためにメッセージを使用します。

メッセージは、メッセージ ID と 0 個以上のパラメータで構成されます。クライアントと Open Server アプリケーションは、各メッセージ ID の意味を同一に解釈するようにプログラムされている必要があります。

ユーザ定義のメッセージ ID は、CS\_USER\_MSGID 以上 CS\_USER\_MAX\_MSGID 以下でなければなりません。SRV\_MINRESMSG から SRV\_MAXRESMSG の間のメッセージ ID は、Sybase の内部使用のために予約されています。

クライアント・アプリケーションは、type を CS\_MSG\_CMD に設定して ct\_command を呼び出し、メッセージを送ります。これにより、Open Server アプリケーションで SRV\_MSG イベントがトリガされます。

## クライアント・データ・ストリーム・メッセージの取得

メッセージ・データ・ストリームは、Open Server アプリケーションの `SRV_MSG` イベント・ハンドラをトリガします。このハンドラはクライアント・メッセージを取得できます。必要な作業は次のとおりです。

- 1 `cmd` を `CS_GET` に設定し、`msgidp` には Open Server がメッセージ ID を格納するバッファへのポインタを指定して、`srv_msg` を呼び出します。

メッセージがパラメータを持つ場合、`srv_msg` は `statusp` パラメータに `SRV_HASPARAMS` を設定します。

[「srv\\_msg」\(351 ページ\)](#) を参照してください。

- 2 必要な場合、パラメータの数を取得するために `srv_numparams` を呼び出します。
- 3 各パラメータを記述および取得するために、`srv_descfmt`、`srv_bind`、`srv_xferdata` を呼び出します。[「パラメータとロー・データの処理」\(152 ページ\)](#) を参照してください。

Open Server アプリケーションで、`SRV_MSG` イベント・ハンドラを使用せずにメッセージを取得することはできません。

## クライアントへのデータ・ストリーム・メッセージの送信

Open Server アプリケーションは、クライアントにメッセージを送信できます。これを実行するために、アプリケーションは次のことを行います。

- 1 `cmd` を `CS_SET` に設定し、`msgidp` にはメッセージ ID が格納されているバッファを指すポインタを指定して、`srv_msg` を呼び出します。

`SRV_HASPARAMS` の `*statusp` 値は、メッセージがパラメータを持つことを意味します。`SRV_NOPARAMS` は、メッセージがパラメータを持たないことを意味します。

[「srv\\_msg」\(351 ページ\)](#) を参照してください。

- 2 `srv_descfmt`、`srv_bind`、`srv_xferdata` を呼び出して各パラメータを記述し、送信します。

Open Server アプリケーションは、SRV\_ATTENTION、SRV\_CONNECT、SRV\_DISCONNECT、SRV\_URGDISCONNECT、SRV\_START の各ハンドラを除く任意のイベント・ハンドラの中からメッセージを送信できます。

## ディレクトリ・サービス

この項では、ディレクトリ・サービスを使用するために Open Server アプリケーションが実行する必要があるタスクについて説明します。説明する内容は、次のとおりです。

- ディレクトリ・ドライバの指定
- ディレクトリへの Open Server アプリケーションの登録

ディレクトリは、情報をディレクトリ・エントリとして保管し、それぞれのエントリに論理名を対応付けます。それぞれのディレクトリ・エントリには、ユーザ、サーバ、またはプリンタなど特定のネットワーク・エンティティについての情報が入っています。ディレクトリ・サービス(ネーミング・サービスと呼ばれる場合もあります)は、ディレクトリ・エントリの作成、修正、取得を管理します。

『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

## ディレクトリ・ドライバの指定

*libtcl.cfg* ファイル内で正しいディレクトリ・サービス・プロバイダが指定されるように編集してあることを確認してから、ディレクトリ・サービスを使用するアプリケーションを実行してください。*libtcl.cfg* ファイルは、`$SYBASE/$SYBASE_OCS/config` ディレクトリ、またはコンテキスト・プロパティ `CS_LIBTCL_CFG` により指定されるパスにあります。サーバ・プロパティ `SRV_DS_PROVIDER` は、*libtcl.cfg* ファイルに指定されているドライバ名を返します。詳細については、各プラットフォームの『Open Client/Server 設定ガイド』を参照してください。`SRV_DS_PROVIDER` プロパティの詳細については、「[srv\\_props](#)」(377 ページ)を参照してください。

使用しているプラットフォームの Open Client/Server でサポートされているディレクトリ・サービスについては、そのプラットフォームの『Open Client/Server 設定ガイド』を参照してください。



## ディレクトリへの Open Server アプリケーションの登録

Open Server アプリケーションは、使用するディレクトリ・サービス・プロバイダを起動時に指定して、アプリケーション自体をディレクトリ・サービスに登録できます。

デフォルト以外のディレクトリ・サービス・プロバイダを指定するには、`srv_props` を使用して `SRV_S_DS_PROVIDER` サーバ・プロパティを設定してください。`SRV_S_DS_PROVIDER` のデフォルト値は、プラットフォームによって異なります。詳細については、使用しているプラットフォームの『Open Client/Server 設定ガイド』を参照してください。

ディレクトリ・サービスに Open Server アプリケーションを登録するには、`srv_props` を使用して、`SRV_S_DS_REGISTER` サーバ・プロパティを `CS_TRUE` (デフォルト) に設定してください。

`SRV_S_DS_REGISTER` を `CS_FALSE` に設定すると、登録できません。

(`cs_ctx_alloc` と `srv_version` を使用して) `CS_CONTEXT` 構造体の割り付けと初期化を行ってから、上記のプロパティを設定し、その後で `srv_init` を呼び出してください。

`srv_init` が呼び出されると、Open Server アプリケーションは次の処理を行います。

- ディレクトリ・サービスから受信するアドレスを取得します。
- `SRV_S_DS_REGISTER` が `CS_TRUE` に設定されている場合は、Open Server アプリケーションのディレクトリ・サービス・エントリを更新するようにディレクトリ・サービスに指示します。
- その結果、ディレクトリ・サービスは、その "currentStatus" 属性を "active" に設定します。

ディレクトリ・サービス・ドライバの初期化に失敗した場合、Open Server は自動的に `interfaces` ファイルをバックアップ・ディレクトリとして使用します。次のいずれかの場合には、`srv_init` 呼び出しが失敗して、指定されたディレクトリ・サービスにアクセスできないことがあります。

- あると仮定していたロケーションに `libtcl.cfg` ファイルがない場合、つまり、そのファイルが読み込み不可能である場合  
情報エラーが返されます。
- あると仮定していたロケーションにディレクトリ・サービス・ドライバがない場合、つまり、そのドライバが読み込み不可能である場合

情報エラーが返されます。

- ディレクトリ・サービスが要求に応答していない場合  
情報エラーが返されます。
- ディレクトリ・サービス内でサーバ・エントリが見つからない場合  
リスナが存在しないことを示すエラーが返されます。この場合、**Open Server** アプリケーションは **interfaces** ファイルをバックアップ・ディレクトリとして使用しません。

## 動的 SQL

動的 SQL を使用すると、クライアント・アプリケーションでは、実行時に値が決定される変数を含む SQL 文を実行することができます。

クライアント・アプリケーションは、プレースホルダを含む SQL 文と識別子を関連付けて、この文を部分的なコンパイルと保存のために **Open Server** アプリケーションに送ることによって、動的 SQL 文を準備します。この状態の文を、*準備文*と呼びます。

クライアント・アプリケーションは、準備文を実行する用意が整うと、SQL 文のプレースホルダに置き換える値を定義し、その文を実行するコマンドを送信します。この値は、コマンドの入力パラメータになります。

定められた回数だけ文が実行されると、クライアント・アプリケーションは文の割り付けを解除します。

## 動的 SQL の利点

動的 SQL によって、クライアント・アプリケーションは、**Open Server** にユーザからのさまざまな情報をさまざまなタイミングでやり取りして対話的に動作できます。これによって **Open Server** アプリケーションは、ユーザが提供したデータを使用して SQL クエリの欠けている部分を補うことができます。

『**Embedded SQL/C プログラマーズ・ガイド**』を参照してください。

## 動的 SQL 要求の処理

クライアントが動的コマンドを発行すると、Open Server は SRV\_DYNAMIC イベントを発生させます。Open Server アプリケーションが動的 SQL の結果を返す場合は、動的 SQL の要求に回答できるように、このアプリケーションに SRV\_DYNAMIC イベント・ハンドラが含まれている必要があります。

### srv\_dynamic ルーチン

SRV\_DYNAMIC イベント・ハンドラ内から、Open Server アプリケーションは srv\_dynamic ルーチンを他の Server-Library ルーチンとともに使用して、クライアントの動的 SQL コマンドを取得し、応答します。「[srv\\_dynamic](#)」(306 ページ)を参照してください。各クライアント・コマンド・タイプ(準備、実行、割り付け解除)は、Open Server アプリケーションからの特定の応答を必要とします。

### コマンド・タイプの検出

SRV\_DYNAMIC イベント・ハンドラ内の最初のタスクは、クライアントが発行した動的コマンドのタイプを取得することです。場合によっては、動的文の ID とテキストも取得します。イベント・ハンドラは、取得した情報を保存して、後でクライアント要求に回答するときにはその情報を参照する必要があります。

### クライアント動的 SQL コマンドへの応答

表 2-19 は、クライアントと Open Server アプリケーション間の動的 SQL 要求と応答の有効なやり取りを示します。右矢印 (→) は、cmd が CS\_GET に設定されている (Open Server アプリケーションはクライアントから情報を取得する)ことを示しています。左矢印 (←) は、cmd が CS\_SET に設定されている (Open Server アプリケーションは情報をクライアントに送信する)ことを示しています。

表 2-19 : 有効な動的 SQL 要求と応答

クライアントのアクション	Open Server アプリケーションの応答
準備要求の発行 (オペレーション・タイプは CS_PREPARE)	→ オペレーション・タイプを取得する。 (srv_dynamic) → 文 ID の長さを取得する。 (srv_dynamic) → 文 ID を取得する。 (srv_dynamic) → 文の長さを取得する。 (srv_dynamic) → 文を取得する。 (srv_dynamic) ← クライアント・コマンドに肯定応答する。 (srv_dynamic) ← 文 ID の長さを送信する。 (srv_dynamic) ← 文 ID を送信する。 (srv_dynamic) ← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)
文の入力パラメータの記述の要求 (オペレーション・タイプは CS_DESCRIBE_INPUT)	→ オペレーション・タイプを取得する。 (srv_dynamic) → 文 ID の長さを取得する。 (srv_dynamic) → 文 ID を取得する。 (srv_dynamic) ← クライアント・コマンドに肯定応答する。 (srv_dynamic) ← 文 ID の長さを送信する。 (srv_dynamic) ← 文 ID を送信する。 (srv_dynamic) ← 入力パラメータのフォーマットを送信する。 (type 引数を SRV_DYNDATA に設定した srv_descfmt と srv_xferdata。アプリケーションはフォーマットを送信 するが、実際のデータは送信しないので、srv_bind の 呼び出しは不要。CS_DATAFMT 構造体の status フィールドについて CS_DESCIN との論理和 (OR) 演 算を実行してから、srv_descfmt を呼び出す必要が ある)。 ← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)

クライアントのアクション	Open Server アプリケーションの応答
<p>文の出力パラメータの記述の要求 (オペレーション・タイプは CS_DESCRIBE_OUTPUT)</p>	<p>→ オペレーション・タイプを取得する。 (<code>srv_dynamic</code>)</p> <p>→ 文 ID の長さを取得する。 (<code>srv_dynamic</code>)</p> <p>→ 文 ID を取得する。 (<code>srv_dynamic</code>)</p> <p>← クライアント・コマンドに肯定応答する。 (<code>srv_dynamic</code>)</p> <p>← 文 ID の長さを送信する。 (<code>srv_dynamic</code>)</p> <p>← 文 ID を送信する。 (<code>srv_dynamic</code>)</p> <p>← 結果ローのフォーマットを送信する。 (<code>type</code> 引数を <code>SRV_DYNDATA</code> に設定した <code>srv_descfmt</code> と <code>srv_xferdata</code>。アプリケーションはフォーマットを送信するが、実際のデータは送信しないので、<code>srv_bind</code> の呼び出しは不要。<code>CS_DATAFMT</code> 構造体の <code>status</code> フィールドについて <code>CS_DESCOUT</code> との論理和 (OR) 演算を実行してから、<code>srv_descfmt</code> を呼び出す必要がある)。</p> <p>← DONE パケットを送信する。 (<code>status</code> 引数を <code>SRV_DONE_FINAL</code> に設定した <code>srv_senddone</code>)</p>

クライアントのアクション	Open Server アプリケーションの応答
<p>実行要求の発行 (オペレーション・タイプは CS_EXECUTE)</p>	<p>→ オペレーション・タイプを取得する。 (srv_dynamic) → 文 ID の長さを取得する。 (srv_dynamic) → 文 ID を取得する。 (srv_dynamic) → 動的パラメータの数を取得する。 (srv_numparams) → 入力パラメータ値を取得する。 (type 引数を SRV_DYNDATA に設定した srv_descfmt、 srv_bind、srv_xferdata) ← クライアント・コマンドに肯定応答する。 (srv_dynamic) ← 文 ID の長さを送信する。 (srv_dynamic) ← 文 ID を送信する。 (srv_dynamic) ← 結果ローを送信する。 (type 引数を SRV_ROWDATA に設定した srv_descfmt、 srv_bind、srv_xferdata) ← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>
<p>即時実行要求の発行 (オペレーション・タイプは CS_EXEC_IMMEDIATE)</p>	<p>→ オペレーション・タイプを取得する。 (srv_dynamic) → 文 ID の長さを取得する。ID の長さは 0。 (srv_dynamic) → 文の長さを取得する。 (srv_dynamic) → 文を取得する。 (srv_dynamic) ← クライアント・コマンドに肯定応答する。 (srv_dynamic) ← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)</p>

クライアントのアクション	Open Server アプリケーションの応答
割り付け解除要求の発行  (オペレーション・タイプは CS_DEALLOC)	→ オペレーション・タイプを取得する。 (srv_dynamic) → 文 ID の長さを取得する。 (srv_dynamic) → 文 ID を取得する。 (srv_dynamic) ← クライアント・コマンドに肯定応答する。 (srv_dynamic) ← 文 ID の長さを送信する。 (srv_dynamic) ← 文 ID を送信する。 (srv_dynamic) ← DONE パケットを送信する。 (status 引数を SRV_DONE_FINAL に設定した srv_senddone)

## 例

サンプル *ctos.c* には、動的 SQL コマンドを処理するコードが記述されています。

## 動的なリスナ

動的なリスナにより、Open Server アプリケーションは `srv_run` ルーチンに対する呼び出しが行われた後に新しいリスナを作成できます。この結果、`interface` ファイル内のエントリが最小限の数で Open Server アプリケーションを起動でき、Open Server アプリケーションはリスナを再起動できます。

## 設定

`SRV_LISTENER_PREBIND` Open Server イベントは、指定された `SRV_PROC` 制御構造体により識別されるリスナを詳細に設定するために使用されます。たとえば、代替の SSL 証明書を指定できる。この設定は、スレッド・プロパティを使用して実行される。

SRV\_LISTENER\_POSTBIND Open Server イベントにより、指定された SRV\_PROC 制御構造体を使用して識別する動的なリスナの最終的な設定を行うことができます。たとえば、スレッド・プロパティを使用してリスナがバインドされるアドレスを決定できます。

### プロパティ

次の2つの Open Server プロパティは、動的なリスナをサポートして使用されます。

- SRV\_S\_NUMLISTENERS プロパティは、クライアント接続の受信に関与する SRV\_PROC 制御構造体の数を返します。
- SRV\_S\_MAXLISTENERS プロパティは、リスナ・スレッドの最大数を制限するために使用されます。

### スレッド・タイプ

SRV\_TLISTENER クライアント・スレッド・タイプは動的リスナに使用されます。

### スレッド・プロパティ

次の3つのスレッド・プロパティは、動的なリスナのサポートに明示的に使用されます。

- SRV\_T\_LISTENADDR プロパティは、指定された SRV\_PROC 制御構造体により識別されたリスナのアドレスを返します。SRV\_PROC 制御構造体がリスナの場合、このプロパティは、リスナが接続を受け入れるアドレスを返します。
- SRV\_T\_LOCALID プロパティは、リスナに使用する SSL 証明書を指定します。
- SRV\_T\_REMOTEADDR プロパティは、SRV\_PROC ピアのアドレスを返します。

### 動的なリスナの起動

動的なリスナは、次のパラメータ値を使用して `srv_spawn` ルーチンを呼び出すことで起動できます。



- *sppp* – スレッド構造体ポインタへのポインタは NULL であることが必要です。
- *stacksize* – スタック・サイズを CS\_UNUSED として指定する必要があります。
- *funcp* – エントリ・ポイント関数ポインタは SRV\_C\_START\_LISTENER であることが必要です。
- *argp* – このパラメータが CS\_TRANADDR 構造体を指し示す必要があります。
- *priority* – 優先度を CS\_UNUSED として指定する必要があります。

## エラー

デフォルトでは、Open Server がエラーに対応する場合は、エラー・メッセージをログ・ファイルに書き込みます。エラー処理ルーチンをインストールすると、開発者はアプリケーションの応答を調整できます。

通常、エラー・ハンドラはエラーのタイプと重大度を検出し、検出した値に基づいて特定の処理を実行します。たとえば、アプリケーションは、*srv\_sendinfo* ルーチンを介してクライアントに特定のエラーを送信し、それ以外のエラーをログ・ファイルに書き込みます。

エラー・ハンドラをインストールするには、*property* 引数を *SRV\_S\_ERRHANDLE* に設定した *srv\_props* ルーチンを使用します。すべてのタイプのエラーを取得するには、アプリケーションで *srv\_version* を呼び出した直後にエラー・ハンドラをインストールする必要があります。エラーが発生すると、Open Server は最も最近インストールされたエラー処理ルーチンを呼び出します。[「\*srv\\_props\*」\(377 ページ\)](#) を参照してください。

エラー・メッセージをトラブルシューティングするには、[「付録 A Open Server エラー・メッセージ」](#) を参照してください。

## エラーのタイプ

Open Server アプリケーション、クライアント・アプリケーション、Open Server 自体のどれもが、Open Server エラーを引き起こす可能性があります。以下に、発生するエラーの各タイプを説明します。

- **Open Server アプリケーション・エラー** — このタイプのエラーは、アプリケーション・コードの誤りが原因で発生します。たとえば、アプリケーションが、最初にデータのフォーマットを記述しないで、クライアントにローのデータを送信しようとする、**Open Server** はエラーを発生させます。
- **クライアント・コマンド・エラー** — このタイプのエラーは、クライアントが送信した要求が不完全な場合や無意味な場合に発生します。クライアントのコードに誤りがあったりネットワークに問題があったりすると、要求は不完全になったり無意味になったりします。**Open Server** アプリケーションは、このようなエラーをクライアント要求のためのイベント・ハンドラで処理します。通常は `srv_sendinfo` を使って該当するエラー・メッセージをクライアントに送ることで、この処理を行います。詳細については、「[クライアント・コマンド・エラー](#)」(38 ページ) を参照してください。また、アプリケーションは、クライアント要求がエラーを引き起こしたことを示すために、`srv_senddone` の `status` パラメータを `SRV_DONE_ERROR` に設定することもできます。
- **Open Server リソース・エラー** — このタイプのエラーは、**Open Server** 自体で発生します。通常はメモリやユーザ接続など、何らかのリソースの不足が原因で発生します。

## エラーの重大度

各 **Open Server** エラーには、番号、重大度レベル、メッセージが関連付けられています。

エラーが発生すると、現在インストールされているエラー・ハンドラ関数が、エラー番号、エラー重大度レベル、メッセージ・テキストを使用して呼び出されます。エラー・ハンドラがインストールされていない場合は、**Open Server** のログ・ファイルにこの情報が記録されます。また、アプリケーションは、`srv_log` を呼び出してログ・ファイルに明示的に書き込むこともできます。

**Open Server** アプリケーションでログ・ファイルの最大サイズを設定するには、`property` 引数を `SRV_S_LOGSIZE` に設定して `srv_props` を使用します。

エラー番号と重大度レベルは、ヘッダ・ファイル `oserror.h` で定義されています。アプリケーションで、定義されているエラー値を使用するには、`oserror.h` をインクルードする必要があります。

表 2-20 に、**Open Server** エラーの重大度レベルを示します。

表 2-20 : エラーの重大度

重大度	意味	適用されるエラーのタイプ
SRV_INFO (10)	情報エラー。ほとんどのエラーがこの重大度である。この重大度レベルは、エラーは起こったが致命的ではないことを示す。通常は、Server-Library 関数の誤った呼び出しによって発生する。たとえば、 <code>srv_descfmt</code> ですべてのカラムを記述しないうちに <code>srv_xferdata</code> を呼び出してローを送ると、SRV_INFO エラーが発生する。	Open Server アプリケーション・エラー  クライアント・コマンド・エラー
SRV_FATAL_PROCESS (15)	致命的なスレッド・エラー。エラーを受信したスレッドには、リカバリ不可能な内部エラーがある。たとえば、アプリケーションが <code>srv_senddone</code> を呼び出さずにイベントから戻ることがある。この重大度のエラーが発生すると、スレッドがクライアント・スレッド、SUB-PROC、またはサイト・ハンドラの場合、Open Server はスレッドに対して SRV_DISCONNECT イベントをキューイングする。次に、Open Server はスレッドを強制終了する。	Open Server リソース・エラー
SRV_FATAL_SERVER (20)	致命的なサーバ・エラー。Open Server は、リカバリ不可能な内部エラーを検出した。このエラーが発生すると、Open Server は Open Server アプリケーションの SRV_STOP イベントをキューイングし、その結果、 <code>srv_run</code> は CS_FAIL を返す。	Open Server リソース・エラー

## オペレーティング・システム・エラー

オペレーティング・システム・エラーが発生した場合、オペレーティング・システム・エラー番号は `SRV_ENO_OS_ERR` 以外の値になります。また、オペレーティング・システム・エラー・テキストにはオペレーティング・システム・エラーの説明が記述されています。たとえば、`srv_init` が `interfaces` ファイルをオープンできない場合、オペレーティング・システム・パーミッション・エラーが原因である可能性があります。

### エラー番号と対応するメッセージ・テキスト

エラー・トークンの完全なリストについては、ヘッダ・ファイルの *oserror.h* を参照してください。対応するエラー・テキストについては、ファイル *oslib.loc* を参照してください。エラー・メッセージをトランスルーディングするには、「[付録 A Open Server エラー・メッセージ](#)」を参照してください。

### 例

すべてのサンプル・プログラムには、Open Server エラー・ハンドラが含まれています。

## イベント

この項では、次の内容について説明します。

- [イベントとは](#)
- [イベント・ハンドラとは](#)
- [標準イベント](#)
- [プログラマ定義イベント](#)
- [例](#)

### イベントとは

Open Server アプリケーションは、クライアントからの要求に応答します。これらの要求には、Server-Library イベントをトリガするものがあります。

すべてのイベントがクライアントのアクティビティによって引き起こされるわけではありません。アプリケーション自体が、ユーザ定義のイベント、および `SRV_DISCONNECT`、`SRV_URGDISCONNECT`、`SRV_STOP` の各イベントをキューイングするため、`srv_event` または `srv_event_deferred` ルーチン呼び出します。`srv_event` ルーチンを使用してイベントを発生させる方法については、関連するページを参照してください。`Open Server` は、致命的なサーバ・エラーに応答すると `SRV_STOP` イベントをトリガすることもできます。`Open Server` では、サーバの開始処理の一部として `SRV_START` イベントが自動的に起動します。

イベントとは特定のコンテキストで発生するもので、特定のカテゴリのアクティビティに対応しています。たとえば、クライアントやリモート・サーバが接続しようとするとき `SRV_CONNECT` イベントがトリガされるのに対して、クライアントからバルク・コピーが要求されると、`Open Server` は `SRV_BULK` イベントを発生させます。

`Open Server` には、標準およびプログラマ定義の2種類のイベントがあります。標準イベントは、`Open Server` 内で定義されます。プログラマ定義イベントは、その名のとおり、アプリケーション内で定義されます。それぞれのイベントの詳細については、「標準イベント」(102 ページ)と「プログラマ定義イベント」(108 ページ)を参照してください。

## イベント・ハンドラとは

イベント・ハンドラとは、イベントを開始する場合に実行する一連のコードのことです。イベントをトリガすると、`Open Server` は、そのイベントとアクティブ・スレッドを実行キューに入れます。次に、そのスレッドは、イベントを処理するルーチンを実行します。このルーチンを「イベント・ハンドラ」と呼びます。

## デフォルト・ハンドラとカスタム・ハンドラ

`Open Server` のデフォルトのイベント・ハンドラ・ルーチンは、個々の標準イベントに対して1つずつあり、さらに全プログラマ定義イベントに対して1つあります。デフォルト・ハンドラは、アプリケーション・プログラマが `srv_handle` ルーチンを使用してインストールするカスタム・イベント・ハンドラのプレースホルダです。デフォルト・ハンドラを使用しないアプリケーションでは、各カスタム・イベント処理ルーチンを定義してインストールしなければなりません。

「`srv_handle`」(335 ページ)を参照してください。

イベント・ハンドラは、動的にインストールすることができます。次のイベント発生時に、新しいイベント・ハンドラが呼び出されません。イベント・ハンドラは、成功した場合には `CS_SUCCEED`、失敗した場合には `CS_FAIL` を返します。現在、Open Server がリターン・コードをチェックする唯一のイベント・ハンドラは、`SRV_START` ハンドラです。`SRV_START` ハンドラから `CS_FAIL` が返されると、`srv_run` は、Open Server を開始しないでアプリケーションに `CS_FAIL` を返します。

### カスタム・ハンドラのコーディング

アプリケーション・プログラマは、イベントへの応答方法を決定し、それによってイベント・ハンドラをコーディングします。イベント・ハンドラには、通常は、イベント・データを処理する呼び出しの標準セットが含まれます。それ以外のコードは、アプリケーション固有のもので、たとえば、パラメータだけでなくメッセージ・テキストも取得できるように、`SRV_MSG` イベント・ハンドラをコーディングします。ただし、特別なメッセージが取得された場合には、`SRV_MSG` イベント・ハンドラにコードを追加してユーザにメールを送信することもできます。

### 標準イベント

表 2-21 に、Open Server の標準イベントと、対応するカスタム・イベント・ハンドラに必要な引数を示します。対応するデフォルト・イベント・ハンドラが実行する関数についても説明しています。

表 2-21 : イベントの説明

イベント	説明	ハンドラに対する引数	デフォルト・イベント・ハンドラ
SRV_ATTENTION	アテンションが受信された。このイベントは通常、クライアントが結果の処理を中断するために <code>ct_cancel</code> を呼び出したときに発生する。 <b>SRV_ATTENTION</b> は即時イベントなので、このイベントが発生すると、 <b>Open Server</b> はクライアントのイベント・キューに追加しないでただちに処理する。 <b>SRV_ATTENTION</b> イベントは割り込みレベルで発生する。	SRV_PROC*	デフォルト・ハンドラは、追加動作を行わない。
SRV_BULK	クライアントがバルク・コピー要求を発行した。	SRV_PROC*	デフォルト・ハンドラは、 <b>"No bulk handler installed"</b> というメッセージをクライアントに送信する。 <b>Open Server</b> はバルク・データを廃棄し、 <b>DONE ERROR</b> をクライアントに返す。
SRV_CONNECT	<b>Client-Library</b> クライアントが <code>ct_connect</code> を呼び出した。	SRV_PROC*	デフォルト・ハンドラは接続を受け付ける。
SRV_CURSOR	クライアントがカーソル要求を送信した。	SRV_PROC*	デフォルト・ハンドラは、 <b>"No SRV_CURSOR handler installed"</b> というメッセージをクライアントに送信する。 <b>Open Server</b> は <b>DONE ERROR</b> をクライアントに返す。

イベント	説明	ハンドラに対する引数	デフォルト・イベント・ハンドラ
SRV_DISCONNECT	<p>クライアント接続を切断する要求が出された。これは、クライアントのサーバからの切断、Open Server の致命的なスレッド・エラー、SRV_STOP イベント、またはクライアントを切断するために明示的にアプリケーション内から出された <code>srv_event</code> の呼び出しによって、トリガされる。</p> <p>Client-Library プログラムは、Open Server からログアウトするため、<code>ct_close</code> または <code>ct_exit</code> を呼び出す。リモート・プロシージャ・コールが完了すると、Adaptive Server Enterprise のリモート接続は終了する。</p>	SRV_PROC*	デフォルト・ハンドラはアクションを起こさない。
SRV_DYNAMIC	クライアントが動的 SQL 要求を送信した。	SRV_PROC*	デフォルト・ハンドラは、"No SRV_DYNAMIC handler installed" というメッセージをクライアントに送信する。Open Server は DONE ERROR をクライアントに返す。
SRV_FULLPASSTHRU	<p>接続に対するネットワーク読み込みが完了した。</p> <p>(このイベントを発生させるため、スレッドの <code>SRV_T_FULLPASSTHRU</code> プロパティを <code>CS_TRUE</code> に設定しておくこと)。</p>	SRV_PROC*	このイベントに対するデフォルト・イベント・ハンドラはない。



イベント	説明	ハンドラに対する引数	デフォルト・イベント・ハンドラ
SRV_LANGUAGE	クライアントから SQL 文などの言語要求が送信された。Client-Library クライアントが言語要求を送信するときは、 <code>ct_command</code> や <code>ct_send</code> が使用される。isql などの対話型クエリ・ツールからも、Open Server アプリケーションに言語要求を送ることができる。	SRV_PROC*	デフォルト・ハンドラは、"No language handler installed" というメッセージと言語要求の最初の数文字をクライアントに送信する。Open Server は DONE ERROR をクライアントに返す。
SRV_LISTENER_PREBIND	指定された SRV_PROC 制御構造体で識別されたリスナの詳細な設定を許可する。たとえば、代替の SSL 証明書を指定できる。この設定は、スレッド・プロパティを使用して実行される。	SRV_PROC*	デフォルト・ハンドラはアクションを起こさない。
SRV_LISTENER_POSTBIND	指定された SRV_PROC 制御構造体で識別されたリスナの最終的な設定を判断できる。たとえば、リスナがバインドされているアドレスを判断できる。設定は、スレッド・プロパティを使用して判断される。	SRV_PROC*	デフォルト・ハンドラはアクションを起こさない。
SRV_MIGRATE_STATE	このイベントは、マイグレーション・ステータスが SRV_MIG_READY または SRV_MIG_FAILED に移行すると必ずトリガされる。移行は、クライアントからマイグレーション・メッセージが送信された結果。 詳細については、「 <a href="#">接続マイグレーション</a> 」(41 ページ)を参照してください。	SRV_PROC*	ステータスが SRV_MIG_READY の場合、デフォルト・ハンドラはアクションを起こさないため、クライアントがマイグレーションを続行できる。ステータスが SRV_MIG_FAILED に変わった場合は、エラーが記録される。

イベント	説明	ハンドラに対する引数	デフォルト・イベント・ハンドラ
SRV_MIGRATE_RESUME	<p>クライアントが結果の待機中に新しいサーバにマイグレートすると、クライアント接続が正常にマイグレートされた後、新しいサーバはSRV_MIGRATE_RESUME イベントを呼び出します。マイグレーション要求が失敗した場合やキャンセルされた場合、イベントはオリジナル・サーバから呼び出されます。</p> <p>詳細については、「<a href="#">接続マイグレーション</a>」(41 ページ)を参照してください。</p>	SRV_PROC*	デフォルト・ハンドラは、最終の完了 (SRV_DONE_FINAL) のみクライアントに送信して、結果を終了する。
SRV_MSG	クライアントがメッセージを送信した。	SRV_PROC*	デフォルト・ハンドラは、"No SRV_MSG handler installed" というメッセージをクライアントに送信する。Open Server は DONE ERROR をクライアントに返す。
SRV_OPTION	クライアントがオプション・コマンドを送信した。	SRV_PROC*	デフォルト・ハンドラは、"No SRV_OPTION handler installed" というメッセージをクライアントに送信する。Open Server は DONE ERROR をクライアントに返す。
SRV_RPC	クライアントまたはリモート Adaptive Server Enterprise がリモート・プロシージャ・コール (RPC) を発行した。	SRV_PROC*	デフォルト・ハンドラは、"RPC < rpcname > received. No remote procedure call handler installed" というメッセージをクライアントに送信する。Open Server は DONE ERROR をクライアントに返す。

イベント	説明	ハンドラに対する引数	デフォルト・イベント・ハンドラ
SRV_START	<p>srv_run を呼び出すと、SRV_START イベントがトリガされる。Open Server アプリケーションは稼働状態。</p> <p>SRV_START イベント・ハンドラは、サーバ・リソースを初期化したり、サービス・スレッドを発生させるのに適している。</p>	SRV_SERVER*	デフォルト・ハンドラはアクションを起こさない。
SRV_STOP	<p>srv_event への呼び出し、または Open Server の致命的サーバ・エラーによって、Open Server アプリケーションを停止する要求がトリガされた。Open Server アプリケーションが停止する。</p> <p>srv_run から返されるものは、アプリケーションが SRV_STOP イベントを要求した場合は CS_SUCCEED、致命的サーバ・エラーによって SRV_STOP イベントが発生した場合は CS_FAIL。</p> <p>このイベントのためのカスタム・ハンドラは、Open Server アプリケーションが終了する前に必要なクリーンアップをすべて実行できる。</p>	SRV_SERVER*	デフォルト・ハンドラはアクションを起こさない。

イベント	説明	ハンドラに対する引数	デフォルト・イベント・ハンドラ
SRV_URGDISCONNECT	<p>このイベントは、Open Server アプリケーションが <code>srv_event</code> を呼び出す場合にのみトリガされる。このイベントに応答して、Open Server は、スレッドの <code>SRV_DISCONNECT</code> ハンドラを呼び出す。Open Server は、スレッド・イベント・キューの最初にこのイベントを登録するので、このイベントは次のイベントとして処理される。</p> <p>キューに入っている他のイベントをバイパスして、即時にスレッドを終了する場合には、アプリケーションがこのイベントを発生させる必要がある。</p> <p><code>SRV_URGDISCONNECT</code> イベントが発生すると、そのスレッドに関連する I/O チャンネルは、<code>dead</code> とマーク付けされる。</p>	SRV_PROC*	デフォルト・ハンドラはアクションを起こさない。

## プログラマ定義イベント

アプリケーションは、`srv_define_event` を使用してプログラマ定義のイベントを定義し、`srv_handle` を使用してインストールします。新しいイベントをクライアントのイベント・キューに登録するには、`srv_event` または `srv_event_deferred` を呼び出す必要があります。

デフォルトのプログラマ定義イベント・ハンドラは、ハンドラがインストールされていないという内容のメッセージをクライアントに送ります。このメッセージには、イベントの番号と名前が含まれています。

プログラマ定義イベントを使用すると、Open Server アプリケーション内の他のスレッドにサービスを提供できます。たとえば、スレッドがトランザクションのログをディスク・ファイルに格納できるようになります。このような機能を設定するには、`srv_define_event` でイベントを定義し、ディスク・ファイルに書き込むハンドラ・ルーチンをインストールして、イベントがキューイングされるサービス・スレッドを作成します。このサービス・スレッドがトランザクション・ロギング・コードを提供します。

## 例

サンプル・プログラム *lang.c* には、簡単な `SRV_LANGUAGE` イベント・ハンドラの例が記述されています。

## ゲートウェイ・アプリケーション

クライアントとしてもサーバとしても動作する Open Server アプリケーションは、「ゲートウェイ・アプリケーション」と呼ばれます。通常、ゲートウェイ・アプリケーションは、クライアントとサーバが直接的に通信できない場合の仲介役として機能します。

たとえば、Open Client アプリケーションは、Oracle データベースのエンジン部分とは直接通信することはできませんが、Oracle データベースへのゲートウェイ・サービスを行う Open Server アプリケーションとは通信できます。この場合、ゲートウェイは、Open Client アプリケーションに対するサーバとして、また Oracle データベースのエンジン部分に対するクライアントとして動作します。

もう1つの例としては、クライアントとリモート Adaptive Server がそれぞれまったく異なるネットワーク上で実行されているために、クライアントがリモート Adaptive Server Enterprise に直接アクセスできない場合が挙げられます。ゲートウェイ・サーバは、クライアントのデータを取得し、パケット化し直してリモート Adaptive Server Enterprise に送信することによって、この問題を解決します。Sybase のミラー・イメージ・クライアントおよびサーバ・ルーチンは、このプロセスを単純化します。サーバ・コンポーネントとクライアント・コンポーネントが、同一のデータ記述構造体を共有することもできます。つまり、ゲートウェイは Server-Library 呼び出しを使用して構造体にリモート・クライアントからの情報を格納し、次に、リモート・サーバと送信するために Client-Library 呼び出しまたは DB-Library 呼び出しを使用して同じ情報を構造体から抽出します。

Adaptive Server Enterprise または Open Server アプリケーションのクライアントとして動作するゲートウェイは、Client-Library または DB-Library ルーチンを使って、クライアントとして機能します。

Open Client アプリケーションのサーバとして動作するゲートウェイは、Server-Library ルーチンを使用して、サーバとして機能します。

---

**警告！** Client-Library は、Open Server アプリケーションの完全な非同期モードでは実行できません。

---

サンプル・プログラム *ctos.c* は、「仮想 Adaptive Server Enterprise」ゲートウェイの一例です。このゲートウェイは、リモート Adaptive Server Enterprise から Sybase クライアントへのデータの渡し方を示します。

---

**警告！** ゲートウェイ・アプリケーションでは、クライアントのルーチンは Open Server のプロセス、つまりスレッドのコンテキスト内で実行されます。この処理（または Open Server アプリケーション全体）が終了する場合、まだ実行途中のクライアント・ルーチンは、予想しない結果を引き起こします。

---

## パススルー・モード

Adaptive Server Enterprise を介して Open Server アプリケーションが Sybase クライアント・アプリケーションに接続するような特殊な場合には、Client-Library と DB-Library は、Open Server が内容に割り込まずにクライアントとサーバ間で TDS パケットを渡せるようにする一連のアプリケーション・プロトコル・パススルー・ルーチンを提供します。この処理は、到着した TDS 情報をアンパックし、送信する前に再びパケット化するよりも効率的です。サンプル `fullpass.c` には、このタイプのゲートウェイの例が記述されています。「[パススルー・モード](#)」(146 ページ) を参照してください。

---

**注意** バージョン 10.0 より前の DB-Library を、Open Server バージョン 10.0 以降を使用するアプリケーションにリンクしないでください。ただし、10.0 以降の Open Server のクライアントとして動作するアプリケーション・プログラムの中で使用することはできます。

---

## 国際化サポート

次に示すように、Open Server は国際化アプリケーションをサポートします。

- Open Server アプリケーションをローカライズできます。  
ローカライズされた Open Server アプリケーションでは、通常、次の処理が行われます。
  - 各国の言語と文字セットでのエラー・メッセージの生成
  - 各国の日時形式の使用
  - 文字列の変換および比較での、特定の文字セットと照合順(「ソート順」ともいう)の使用
- ローカライズされたクライアントをサポートします。

ローカライズされたクライアントでは、ロケールに応じた言語、日時形式、文字セットが使用されます。これらは、Open Server アプリケーションの言語、日時形式、文字セットとは異なる場合があります。ローカライズされたクライアントをサポートするには、Open Server アプリケーションは入力データをアプリケーション自体の言語と文字セットに変換するだけでなく、出力メッセージやデータをクライアントの言語と文字セットに変換しなければなりません。

ここでは、次の内容について説明します。

- Open Server アプリケーションのローカライゼーション
- ローカライズされたクライアントのサポート
- ローカライゼーションに関連するクライアント要求
- ローカライゼーション・プロパティ
- ローカライゼーションのプログラム例

Open Client/Server のローカライゼーションについては、『Open Client/Server 開発者用国際化ガイド』に記載されています。このマニュアルを参照して、Server-Library のローカライゼーション・メカニズムとローカライゼーションに環境変数がどのように影響するかを理解してください。

使用しているプラットフォームのローカライゼーションに関する情報については、『Open Client/Server 設定ガイド』を参照してください。

## Open Server アプリケーションのローカライゼーション

Open Server アプリケーションのローカライゼーションによって、次の内容が決まります。

- エラー・メッセージを生成する言語と文字セット

---

**注意** `SRV_S_USESRVLANG` プロパティと `SRV_T_USESRVLANG` プロパティを使用して指定された言語は、エラー・メッセージ生成時に、サーバの言語よりも優先されます。

---

- すべてのデータ操作に使用される文字セットと照合順

Open Server アプリケーションは初期ローカライゼーション値やカスタム・ローカライゼーション値、またはその両方を使用することもできます。



通常、国際化された Open Server アプリケーションは、LC\_ALL および LANG 環境変数、またはロケール・ファイル内の「デフォルト」のエントリによって特定された初期ローカライゼーション値を使ってローカライズが行われます。

初期ローカライゼーション値は実行時に確定されます。このとき、Open Server アプリケーションが CS-Library ルーチン `cs_ctx_alloc` を呼び出して `CS_CONTEXT` 構造体を割り付けます。アプリケーションがこの呼び出しをする場合、CS-Library は最初のローカライゼーション情報を新しいコンテキスト構造体にロードします。

初期ローカライゼーション値がアプリケーションの要求を満たさない場合、アプリケーションでは、`CS_LOCALE` 構造体を使用してコンテキスト構造体にカスタム・ローカライゼーション値を設定できます。「[CS\\_LOCALE 構造体を使用したカスタム・ローカライゼーション値の設定](#)」(114 ページ)を参照してください。

## ローカライズされたクライアントのサポート

Open Server アプリケーションによっては、ローカライズされたクライアントに対する最初のローカライゼーション値で十分な場合もあります。このような Open Server アプリケーションでは、ローカライズされたクライアントをサポートするために、それ以上の手順は必要ありません。

しかし、ローカライズされたクライアントに対して追加のサポートが必要な Open Server アプリケーションもあります。特に、次のような場合は、Open Server アプリケーションには、ローカライズされたクライアントをサポートするための追加の手順が必要です。

- CS-Library エラー・メッセージをクライアントに送り返す場合  
この場合、Open Server アプリケーションは、CS-Library がクライアントの言語と Open Server アプリケーションの文字セットでメッセージを生成していることを確認する必要があります。  
この実行方法の詳細については、「[クライアントに対する CS-Library メッセージのローカライズ](#)」(115 ページ)を参照してください。
- ゲートウェイとして機能する場合  
この場合、Open Server アプリケーションは、リモート・サーバへの接続がクライアントの言語と Open Server アプリケーションの文字セットを使用していることを確認する必要があります。

この実行方法の詳細については、「[ゲートウェイ・アプリケーションに対するローカライズされた接続の作成](#)」(116 ページ)を参照してください。

- クライアント・アプリケーションが言語または文字セットの変更を要求した場合

この場合、Open Server アプリケーションは、クライアント・スレッドに対する言語または文字セットを変更する必要があります。

この実行方法の詳細については、「[言語と文字セットの変更要求](#)」(117 ページ)を参照してください。

## CS\_LOCALE 構造体を使用したカスタム・ローカライゼーション値の設定

クライアントが Open Server アプリケーションに接続するときに、Open Server は、クライアントの言語と文字セットを反映する CS\_LOCALE 構造体を作成します。たとえば、`frch/cp850` のクライアントが `us_english/iso_1/binary` の Open Server アプリケーションにログインする場合、Open Server アプリケーションはこの接続に対する `frch/cp850` の CS\_LOCALE 構造体を作成します。

Open Server プログラマは、`cs_locale` を呼び出して、新しく割り当てた CS\_LOCALE 構造体にこの情報をコピーし、構造体内の情報を使用できます。

`srv_version` を呼び出す前に、アプリケーションワイドなコンテキスト構造体にカスタム・ローカライゼーション情報をインストールできます。このために、アプリケーションは次のことを行います。

- 1 `cs_loc_alloc` を呼び出して CS\_LOCALE 構造体を割り付けます。
- 2 `type` を `CS_LC_ALL` に設定して `cs_locale` を呼び出し、CS\_LOCALE にカスタム・ローカライゼーション値をロードします。`type` が `CS_LC_ALL` に設定されると、内部的に一貫性のあるローカライゼーション値が CS\_LOCALE にロードされます。
- 3 `property` を `CS_LOC_PROP` に設定して `cs_config` を呼び出し、カスタム・ローカライゼーション値をアプリケーションのコンテキスト構造体にコピーします。
- 4 `cs_loc_drop` を呼び出して、CS\_LOCALE の割り付けを解除します。

## クライアントに対する CS-Library メッセージのローカライズ

Open Server アプリケーションが、アプリケーション自体のコンテキスト構造体をパラメータとして使用して CS-Library ルーチン呼び出すと、呼び出しの結果として CS-Library が生成するエラー・メッセージでは、Open Server アプリケーションの言語と文字セットが使用されません。

たとえば、`cs_convert` 呼び出しのコンテキスト・パラメータが `us_english/iso_1` を示している場合、`cs_convert` 呼び出しが失敗すると、CS-Library は `us_english/iso_1` のメッセージを生成します。

---

**注意** CS-Library ルーチンのパラメータとして `CS_LOCALE` 構造体が指定されると、この構造体の中のローライゼーション値によってコンテキスト・パラメータの中のローライゼーション値が上書きされます。

---

Open Server アプリケーションの言語と文字セットで CS-Library メッセージを取得できるのは、Open Server アプリケーションが CS-Library メッセージのログを取るか、または CS-Library メッセージを保持する場合のみです。

しかし、Open Server アプリケーションが CS-Library のエラー・メッセージをクライアントに送り返す場合、CS-Library はクライアントの言語と Open Server アプリケーションの文字セットでメッセージを生成します。

メッセージは、クライアントが理解できるクライアントの言語で生成される必要があります。

次の2つの理由から、メッセージは Open Server アプリケーションの文字セットでなければなりません。

- Open Server アプリケーションは、通常は、すべてのメッセージをログ・ファイルに記録します。したがって、ログに記録されたすべてのメッセージが同一の文字セットを使用することが重要です。
- Open Server は、メッセージなどの出力データに対して自動的に文字セット変換を行います。Open Server の文字セットで生成されたメッセージは、クライアントの文字セットに正しく変換されます。

アプリケーションは、クライアント・スレッドごとに正しくローカライズされた `CS_CONTEXT` 構造体を設定することによって、メッセージを正しい言語および文字セットで確実に生成します。また、クライアントの代わりに CS-Library ルーチン呼び出すときに、これらの `CS_CONTEXT` 構造体を使用できます。

CS\_CONTEXT 構造体のローカライズ方法については、「[CS\\_CONTEXT 構造体のローカライズ](#)」(116 ページ)を参照してください。

## ゲートウェイ・アプリケーションに対するローカライズされた接続の作成

Open Server アプリケーションがゲートウェイとして機能している場合は、クライアントの言語と Open Server の文字セットを使用してリモート・サーバに接続されるようにする必要があります。

---

**注意** Open Server の文字セットは、リモート・サーバの文字セットと同じである必要はありませんが、リモート・サーバがそのサーバ自体の文字セットに変換可能なものである必要があります。

Adaptive Server Enterprise は、2つの西欧文字セット間での変換や2つの日本語文字セット間での変換はできますが、西欧文字セットから日本語文字セットへの変換はできません(また、その逆の日本語文字セットから西欧文字セットへの変換もできません)。

たとえば Adaptive Server Enterprise は、ともに西欧言語グループの文字セットである ISO 8859-1 と CP850 との間では変換できますが、西欧言語グループの文字セットである ISO 8859-1 と、東欧言語グループの文字セットである CP 1250 との間での変換は実行できません。

Open Server では、同じ言語グループの文字セットであるかどうかに関係なく、サポートされている2つの文字セット間での変換ができます。ただし、異なる言語グループの文字セット間での変換を行う場合は、アルファベット以外の文字は失われる可能性があります。

---

このためのアプリケーションにとって一番簡単な方法は、クライアント接続ごとに正しくローカライズされた CS\_CONTEXT 構造体を設定し、ローカライズされたコンテキストのもとでクライアントに対するリモート接続を割り当てることです。

CS\_CONTEXT 構造体のローカライズ方法については、次の「[CS\\_CONTEXT 構造体のローカライズ](#)」を参照してください。

## CS\_CONTEXT 構造体のローカライズ

Open Server アプリケーションで、クライアント・スレッドに対する CS\_CONTEXT 構造体を正しくローカライズするには、次の手順に従います。

- 1 `cs_ctx_alloc` を呼び出し、クライアント・スレッドに対する `CS_CONTEXT` を割り付けます。
- 2 `cs_loc_alloc` を呼び出し、`CS_LOCALE` 構造体を割り付けます。
- 3 `srv_thread_props` を呼び出し、クライアント・スレッドの既存の `CS_LOCALE` 構造体をコピーします。これによって、新しい `CS_LOCALE` はクライアントの言語と文字セットで設定されます。
- 4 `type` を `CS_SYB_CHARSET` に設定して `cs_locale` を呼び出し、クライアントの文字セットを `Open Server` の文字セットに置き換えます。
- 5 `property` を `CS_LOC_PROP` に設定して `cs_config` を呼び出し、ローカライゼーション情報を `CS_LOCALE` から `CS_CONTEXT` にコピーします。
- 6 必要に応じて、`cs_loc_drop` を呼び出し、`CS_LOCALE` の割り付けを解除します。アプリケーションは、必要に応じて `cs_locale` を呼び出し、`CS_LOCALE` 構造体を再利用してローカライゼーション値を変更できます。

## クライアント要求に対する応答

クライアントは、次のことを要求できます。

- 言語と文字セットの変更
- ローカライゼーション情報の要求

## 言語と文字セットの変更要求

クライアントは、`Open Server` に接続するときに、ログイン・レコード内で言語と文字セットを指定します。`Open Server` は、この情報を使用して、クライアント・スレッドに対する `CS_LOCALE` と「文字セット変換ルーチン」を設定します。

`Open Server` は、自動的にこの処理を行います。`Open Server` アプリケーションは、ログイン時にローカライズされたクライアントを処理する必要はありません。

ただし、ログインした後も、クライアントは言語と文字セットを変更できます。クライアントから言語または文字セットの変更要求が送信されると、`Open Server` アプリケーションはクライアント・スレッドの `CS_LOCALE` 構造体で要求された変更を実行します。

クライアントは、次の2つの方法で言語または文字セットの変更を要求できます。

- `ct_command` を使用して送信される、言語に基づいたオプション・コマンドを使用する。このタイプのコマンドは `SRV_LANGUAGE` イベントをトリガします。その結果、Open Server アプリケーションは `SRV_LANGUAGE` イベント・ハンドラの内部で要求を処理します。
- `ct_options` を使用して送信される、オプション・コマンドを使用する。このタイプのコマンドは `SRV_OPTION` イベントをトリガします。その結果、Open Server アプリケーションは `SRV_OPTION` イベント・ハンドラの内部で要求を処理します。

どちらの場合も、Open Server アプリケーションは次の方法で応答します。

- 1 新しい言語または文字セットを使用して、`CS_LOCALE` 構造体を設定します。
- 2 `property` を `SRV_T_LOCALE` に設定して `srv_thread_props` ルーチンを呼び出し、スレッド接続の言語または文字セットを変更します。

表 2-22 は、クライアント・スレッドの言語または文字セットを変更する方法を示します。

表 2-22 : 言語または文字セットの変更

手順	アプリケーションの手順	目的	説明
1	<code>cs_loc_alloc</code> を呼び出す。	<code>CS_LOCALE</code> 構造体を割り付ける。	この呼び出しは、Open Server アプリケーション・コンテキストの現在のローカライゼーション情報を新しい <code>CS_LOCALE</code> 構造体にコピーする。
2	<code>property</code> を <code>SRV_T_LOCALE</code> に設定して <code>srv_thread_props(GET)</code> を呼び出す。	クライアント・スレッドの既存のローカライゼーション値を新しい <code>CS_LOCALE</code> 構造体にコピーする。	
3	<code>cs_locale</code> を呼び出す。	要求された言語または文字セットを使用して <code>CS_LOCALE</code> 構造体を上書きする。	「 <a href="#">CS_CONTEXT 構造体のローカライズ</a> 」(116 ページ)を参照してください。

手順	アプリケーションの手順	目的	説明
4	property を SRV_T_LOCALE に設定して <code>srv_thread_props(SET)</code> を呼び出す。	新しい言語または文字セットを使用して、クライアント・スレッドを設定する。	
5	オプションで <code>cs_loc_drop</code> を呼び出す。	CS_LOCALE 構造体の割り付けを解除する。	アプリケーションは、CS_LOCALE 構造体の割り付けを解除する前に構造体を再使用できる。 構造体のローカライゼーション値を再利用する前に、必要に応じてアプリケーションは <code>cs_locale</code> を呼び出して、この値を変更できる。

## ローカライゼーション情報の要求

ログインした後、クライアントは次の情報を要求できます。

- サーバの文字セットの名前
- サーバのソート順の名前
- クライアント文字セットのための文字セットの定義
- クライアントのソート順に対するソート順定義

クライアントは、RPC コマンドを使用し、`sp_serverinfo` システム・レジスタード・プロシージャを介してこれらの情報を要求します。

Open Server からの応答として、`sp_serverinfo` システム・レジスタード・プロシージャを介して要求された情報が自動的に送り返されます。このとき、Open Server アプリケーションは何も行う必要はありません。実際には、Open Server アプリケーションは、要求が送られたことを認識していません。

「レジスタード・プロシージャ」(185 ページ) を参照してください。

## ローカライゼーション・プロパティ

ローカライゼーションには2つのプロパティが関係します。

- `SRV_S_USESRVLANG`
- `SRV_T_USESRVLANG`

これらのプロパティは、Open Server がエラー・メッセージを Open Server アプリケーションの言語で生成するかクライアントの言語で生成するかを指定します。

SRV\_S\_USESRVLANG はサーバワイドなプロパティであり、`srv_props` によって設定されます。この値は、SRV\_T\_USESRVLANG のデフォルト値として使用されます。

SRV\_T\_USESRVLANG はスレッドのプロパティであり、`srv_thread_props` によって設定されます。新しいスレッド構造体が割り当てられたとき、SRV\_T\_USESRVLANG は SRV\_S\_USESRVLANG からデフォルト値を抽出します。

SRV\_T\_USESRVLANG が CS\_TRUE の場合、Open Server はエラー・メッセージをサーバの言語で生成します。

SRV\_T\_USESRVLANG が CS\_FALSE の場合、Open Server はエラー・メッセージをクライアントの言語で生成します。

[「プロパティ」 \(157 ページ\)](#) を参照してください。

## ローカライゼーションの例

サンプル `ctos.c` には、CS\_LOCALE 構造体のカスタマイズ方法の一例が記述されています。サンプル `intlchar.c` は、文字セットと各国言語の設定、およびクエリを処理します。

## 言語呼び出し

Open Server は、言語イベントを柔軟に処理する機能を備えています。クライアント・アプリケーションが、`type` 引数を CS\_LANG\_CMD に設定した `ct_command` を使って情報を送ると、SRV\_LANGUAGE イベントがトリガされます。RPC ストリームが名前とパラメータという個別の要素から構成されているのに対して、言語情報は区別のない文字ストリームとして着信します。SRV\_LANGUAGE イベント・ハンドラには、その文字ストリームを有効なコンポーネントに解析できるコードが必要がります。SQL クエリは、言語ストリームの一例です。



これは、自然言語で入力を受け入れるアプリケーションにとって有効な機能です。たとえば、SQL データベースに対してユーザが英語でクエリを実行できる衣料店アプリケーションがあり、営業員が「How many shirts in blue?」という質問を入力したとします。フロントエンドクライアントアプリケーションは、`ct_command` を呼び出してこの自然言語クエリを Open Server ゲートウェイアプリケーションに送信します。SRV\_LANGUAGE ハンドラは、このテキストを解析して下記の Transact-SQL クエリを作成し、クエリをリモート・データベースに送信します。

```
select quantity from inventory_tab where color = "blue" and type = "shirt"
```

SRV\_LANGUAGE イベント・ハンドラは、次の手順で言語データを処理する必要があります。

- 1 `srv_langlen` を呼び出して、言語要求バッファの長さを取得します。
- 2 `srv_langlen` から返された長さに NULL 終了バイト分の 1 バイトを加算した大きさを持つ、ローカル・アプリケーション・バッファを割り付けます。
- 3 `srv_langcpy` を呼び出し、要求データの全部または一部をローカル・バッファにコピーします。
- 4 ローカル・バッファの内容を処理します。

## ログイン・リダイレクトと拡張 HA フェールオーバーのサポート

ログイン・リダイレクトと拡張 HA フェールオーバーのサポートにより、サーバのクラスタで、すべての着信クライアント接続を対象に負荷のバランスをとることができます。

この機能は、次の 3 つのルーチン、`srv_send_ctlinfo`、`srv_getserverbyname`、`srv_freeserveraddrs` によってサポートされます。

`srv_send_ctlinfo` ルーチンは、ログイン・リダイレクトと拡張 HA フェールオーバーの両方をサポートしますが、`srv_getserverbyname` と `srv_freeserveraddrs` は、Open Server アプリケーションでサーバ名を接続情報に変換できるようにします。これらのルーチンの詳細については、「[srv\\_send\\_ctlinfo](#)」(431 ページ)、「[srv\\_getserverbyname](#)」(334 ページ)、「[srv\\_freeserveraddrs](#)」(319 ページ)を参照してください。

次のプロパティがこれらのルーチンをサポートします。

- **SRV\_S\_HASERVER** は読み込み専用のサーバ・プロパティで、`interfaces` ファイルから **HAFILOVER** 値を返します。返される値は、`srv_init` によって設定されたサーバ名に対応します。
- **SRV\_T\_REDIRECT** は読み込み専用のスレッド・プロパティで、ログイン・レコード内の **TDS\_HA\_LOG\_REDIRECT** ビットの設定を返します。
- **SRV\_T\_HA** はスレッド・プロパティで、ログイン・レコードから HA 関連情報の設定を **CS\_INT** ビットマスクとして返します。提供される情報には、セッション (**SRV\_HA\_LOGIN**)、フェールオーバー (**SRV\_HA\_LOGIN\_FAILOVER**)、再開 (**SRV\_HA\_LOGIN\_RESUME**) の各ビットが含まれます。
- **CS\_SESSIONID** は、セッション ID を保持する型定義です。
- **SRV\_T\_SESSIONID** は、クライアントが Open Server に送信するセッション ID をログイン・レコードで返します。  
**SRV\_T\_SESSIONID** を使用しても、**SRV\_CONNECT** ハンドラでセッション ID をクライアントに送信できます。「[クライアントに対する異なるサーバへのマイグレーションの指示](#)」(50 ページ)を参照してください。
- **SRV\_NEG\_SESSIONID** は、`srv_negotiate` の語法でネゴシエートされたログイン情報の一種で、クライアントのセッション ID 情報の送信をサポートします。

## メッセージ

Open Server では、次の 3 つのタイプのメッセージがあります。

- データ・ストリーム・メッセージ - クライアントおよびサーバは、データ・ストリーム・メッセージを使用して情報を交換できます。「[データ・ストリーム・メッセージ](#)」(86 ページ)を参照してください。
- スレッド・メッセージ - スレッドは、スレッド・メッセージを使用して情報を交換できます。「[マルチスレッド・プログラミング](#)」(123 ページ)を参照してください。

- エラー・メッセージ - Open Server は、エラー・メッセージを使用してエラー状態を通知します。「[エラー](#)」(97 ページ) を参照してください。

## マルチスレッド・プログラミング

Open Server は、マルチスレッド・アーキテクチャを使用します。マルチスレッド・サーバ・アプリケーションは、「スレッド」の集まりとして動作し、それぞれのスレッドが独自のタスクを達成するためにルーチンを実行します。

### スレッドについて

スレッドは、Open Server アプリケーションを通じた、特定の実行の道筋と考えることができます。各クライアントは、スレッドを使用して接続を管理し、イベント・ハンドラとプロシージャを呼び出して要求に応答します。Open Server ランタイム・システムには、メッセージの配信、サーバ対サーバ通信の処理、タスクのスケジュールなどのサーバ・アクティビティを管理するスレッドが複数あります。アプリケーションは、他のアプリケーション固有のアクティビティのためにサーバ・スレッドを生成できます。

マルチスレッド・システムとしての Open Server アプリケーションでは、スレッドが実行するさまざまなアクティビティのスケジューリング、共有リソースへのスレッドのアクセスのネゴシエーション、スレッドが相互通信するための手段の提供を行う必要があります。「[スケジューリング](#)」(127 ページ) および「[ツールと手法](#)」(130 ページ) を参照してください。

### スレッドのタイプ

Open Server は、次の4種類のスレッド、*preemptive*、*event-driven*、*service*、*site-handler* を使用します。

## プリエンプティブ・スレッド

Open Server バージョン 12.5 以上では、すべてのプラットフォームに対してプリエンプティブ・スレッドが導入されました。スレッド・ライブラリを使ってアプリケーションを作成する場合の事前の注意点について以下に説明します。

## スレッドセーフな関数

アプリケーションがリエントラントであることを確認するため、次の点を確認してください。

- 提供される C ライブラリ関数のリエントラント・バージョンを使用している。
- 非リエントラント C (またはその他の) ライブラリ関数を安全な方法で使用している。
- ミューテックス・ロック、またはその他のロックを使用して、グローバル変数と共有構造体を保護している。
- アプリケーションで、静的バッファを指すポインタを返す関数を使用していない。
- 正しいプロセッサ・フラグとリンカ・ディレクティブを使用してコンパイルを実行する。

---

**注意** ある UNIX システムでリエントラントである C ライブラリ関数が、他の UNIX システムでもリエントラントであるとは限りません。ご使用のプラットフォームの移植に関するガイドを参照して、C 関数がリエントラントであるかどうかを確認してください。

---

## スレッドセーフ・コードとプリエンプティブ・モード

一度に複数の Open Server スレッドを実行できます。また、1つのスレッドを他のスレッドのためにプリエンプティブにできます。この操作によって同時実行性が向上します。特に、SMP システムの場合に効果があります。ただし、コードがスレッドセーフである必要があります。これは Open Server コード、ユーザのイベント・ハンドラ、コールバック関数に適用されます。

## SRV\_S\_PREEMPT の動作

SRV\_S\_PREEMPT を CS\_TRUE に設定した場合、複数の Open Server スレッドを同時に実行できます。また、オペレーティング・システムにより、これらのスレッドは相互にプリエンティブになります。これらのスレッドは、バインドされません。

SRV\_S\_PREEMPT が CS\_FALSE に設定されている場合は、ある Open Server スレッドの実行に割り込みをかけて別の Open Server スレッドを実行することはできません。また、2つの Open Server スレッドを同時に実行することもできません。

また、スレッド・ライブラリとともに使用する場合、SRV\_S\_PREEMPT を CS\_TRUE または CS\_FALSE のいずれに設定しても、SRV\_S\_CURTHREAD の一部の関数は無効になります。これは、SRV\_S\_PREEMPT の設定にかかわらず、スレッド・ライブラリがシグナル処理スレッドにより処理されるシグナルを使用するためです。

Open Server スレッドの実行が再開されると、1つのミューテックスが有効化されます。SRV\_C\_SUSPEND コールバックの実行後、Open Server が特定のタスクを実行できる状態になったときに、ミューテックスが解放されます。この場合、サーバワイドのミューテックスは1つだけ存在します。

オペレーティング・システムがこのようなスレッドを再開しても、SRV\_C\_RESUME と SRV\_C\_SUSPEND コールバック関数は呼び出されません。これらの関数は、特定の Open Server スレッドが実行を停止または再開したときにのみ呼び出されます。たとえば、ユーザ Open Server スレッドの言語要求が着信し、言語イベント・ハンドラの実行後、このスレッドがスリープ状態になる前の時点などです。

## 実装固有の事項

ほとんどの UNIX プラットフォームでは、スレッドは POSIX スレッドに基づいており、バインド解除されています。HP および Linux では、スレッドはバインドされています。Windows では、スレッドは Win32 スレッドです。

プラットフォームでのスレッドの使用方法の詳細については、ご使用のプラットフォームに付属の各ベンダーのマニュアルを参照してください。

## イベント駆動型スレッド

クライアント接続を制御するスレッドは、イベント駆動型です。アクションを必要とする要求は、サーバ・イベントをトリガします。イベントの詳細については、「[イベント](#)」(100 ページ)を参照してください。

クライアント・イベントが発生すると、そのイベントは **Open Server** によってスレッドのイベント・キューに登録されます。スレッドの次回実行時には、イベント・キューから次のイベント要求を読み込みます。**Open Server** は、そのイベントに関連しているイベント・ハンドラを呼び出します。ハンドラが復帰すると、スレッドはキューにある次のイベントを読み込もうとします。イベントがない場合は、スレッドは「スリープ」します。

たとえば、クライアント・アプリケーションがサーバにログインしようとするときには、ランタイム・システムは接続を処理するスレッドを作成し、そのスレッドのキューに **SRV\_CONNECT** イベントに登録します。このスレッドの実行時には、**SRV\_CONNECT** イベントを処理するためにインストールされているルーチンが実行されます。デフォルト・ハンドラは、単に接続を受諾するだけです。

**SRV\_CONNECT** のカスタム・ハンドラをインストールすることもできます。この場合は、**ログイン名**とパスワードをチェックして両方とも有効なときにユーザのログインを許可します。

イベント駆動型のスレッドは主にクライアント要求を処理するために存在しますが、サーバ内でサービス・ルーチンを実行するためにはユーザ定義イベントと併せて使用することもできます。

## サービス・スレッド

どのクライアント接続からも独立して動作する **Open Server** スレッドを作成できます。このようなスレッドがサービス・スレッドと呼ばれるのは、実行するルーチンが、通常は、イベント駆動型のクライアント・スレッドに対してサービスを提供するからです。クライアント・スレッドとは違って、サービス・スレッドはイベントによって起動しません。その代わりに、スレッド作成時にスレッド用の実行ルーチンを設定します。このルーチンは、サーバによってすぐに実行キューに登録されます。あるルーチンを実行するために作成されたサービス・スレッドは、そのルーチンが戻ったところで、消滅します。

アプリケーションは、Open Server でさまざまなタスクを実行するためにサービス・スレッドを使用できます。実際、Open Server ランタイム・システムは、サーバ管理ルーチンを実行するサービス・スレッドから構成されています。サービス・スレッドは、クライアント I/O、つまりクライアント・コマンドを読み込んで結果を返すことはできません。

Open Server は、イベントがトリガされたときに動作させるイベント・コードをスケジューリングします。それとは対照的に、アプリケーションは `srv_wakeup`、`srv_sleep`、`srv_yield` ルーチンを使って明示的にサービス・スレッド・コードをスケジューリングする必要があります。さらに、メッセージ・キューのスケジューリングについては、プリエンプティブ・モードで動作していないときに行う必要があります。

## サイト・ハンドラ・スレッド

Adaptive Server Enterprise が Open Server アプリケーションに接続すると、Open Server はサイト・ハンドラ・スレッドを作成します。

Open Server アプリケーションがサーバ対サーバ RPC を受け取ると、Open Server は SUB-PROC を作成します。SUB-PROC は、サーバ対サーバ RPC が完了すると消えます。サイト・ハンドラ・スレッドは、Adaptive Server Enterprise が Open Server アプリケーションへの接続をクローズすると消えます。

Open Server アプリケーションがサイト・ハンドラ・スレッドにアクセスするのは、`SRV_CONNECT` または `SRV_DISCONNECT` イベント・ハンドラ内だけです。このイベント・ハンドラ以外では、サイト・ハンドラ・スレッドは完全に内部的に使用されます。

## スケジューリング

Open Server は、実行中のスレッドを定期的に中断して他のスレッドを再開する方法で、同時実行性を実現します。このコンテキスト切り替えは頻繁にしかも迅速に行われるため、Open Server クライアントからはスレッドが連続的に動作しているように見えます。

**スケジューラ**は、コンテキスト切り替えを実行するランタイム・システムのスレッドです。スレッドに、スタックとマシン・レジスタ環境を含む実行コンテキストがあります。スケジューラは、実行中のスレッドの実行コンテキストを保存し、再開するスレッドを選択し、保存したコンテキストをリカバリしてから、コンテキストを実行します。スケジューラは透過的に動作しますが、**Open Server** コードを書くには、次のことを理解しておく必要があります。

- スケジューラがどのように呼び出されるか (スケジューリング方法)
- スケジューラが再開するスレッドをどのように選択するか

### スケジューリング方法

スケジューリング方法によって、1つの実行中スレッドから別のスレッドに制御が移行される時期を決定します。**Open Server** アプリケーションは、次の2つのスケジューリング方法、**非プリエンプティブ**または**プリエンプティブ**のいずれか1つを使用します。デフォルトでは、非プリエンプティブです。ほとんどのプラットフォームでは非プリエンプティブの方法しか使用できません。

### 非プリエンプティブ・スケジューリング

非プリエンプティブ・スケジューリングでは、コンテキストの切り替えを予測できます。次に示す状況でのみ切り替えが行われます。

- スレッドが、ネットワーク I/O を行っている **Server-Library** または **Client-Library** ルーチン呼び出す。

スレッドがネットワーク接続に対して読み出しまたは書き込みを行った場合は、ランタイム・システムは読み込みまたは書き込みの完了を待っているスレッドの実行を中断します。ネットワーク I/O は比較的遅いため、サーバは I/O が完了する間に他のスレッドを実行することによって、時間をより有効に使用することができます。

- スレッドが、実行が再開されるのを待機してスリープする。

たとえば、他のスレッドが共有メモリ内のデータ・オブジェクトを更新している場合、そのオブジェクトにアクセスするには、スレッドは更新が終了するまで待機している必要があります。アプリケーションが次のルーチン呼び出すと、スレッドはスリープします。

- `srv_sleep`



- `srv_getmsgq (SRV_M_WAIT)` または `srv_lockmutex`  
のように、要求されたリソースを待機している間はスレッドがスリープする `Server-Library` ルーチン
- スレッドがそれ自体を故意に中断させ、他のスレッドを実行可能にするために `srv_yield` を呼び出す。そのスレッドは実行可能のままであり、`srv_yield` 呼び出し後の文でオペレーションを再開する。スリープしない、またはネットワーク I/O を実行しない、処理時間の長いルーチンを作成する場合は、ルーチンがサーバを独占しないように定期的に `srv_yield` を呼び出してください。

### プリエンプティブ・スケジューリング

プリエンプティブ・スケジューリングでは、上記のイベントのいずれかが起きた場合、また、システムが実行中のスレッドに割り込むときに、コンテキスト切り替えが起こる可能性があります。プリエンプティブ・スケジューリングはオペレーティング・システムのスレッド管理機能に依存しているので、システムが開始する切り替えは、予測できません。通常、オペレーティング・システムには、確実にスレッド間で効率的な時間配分を行う複雑なアルゴリズムがあります。

`property` を `SRV_S_PREEMPT` に設定した `srv_props` ルーチンを使用することによって、プリエンプティブ・スケジューリングを選択できます。プリエンプティブ・スケジューリングは、すべてのプラットフォームで使用できるわけではありません。使用しているアプリケーションのプラットフォームでプリエンプティブ・スケジューリングが使用可能かどうかを確認するには、`srv_capability` を呼び出します。

### 再開するスレッドの選択

`Open Server` は、複数の実行キューを保持します。実行キューとは、中断されているがスリープはしていないスレッドのリストです。各キューには、同じ実行優先順位を持つスレッドがあります。スケジューラは、最高優先順位のキューの中の、最も長い期間そのキューにあったスレッドを再開させます。通常は、スレッドは同じ優先順位で実行されるので、この選択方法では先入れ先出しを基本的に実行時間を割り当てます。

スレッドの優先順位を調節して、他のスレッドより先にスケジューラがまずそのスレッドを実行するようにしたり、他に実行するスレッドがない場合のみそのスレッドが実行されるようにすることができます。たとえば、リアルタイム・データを読み込むスレッドについては、処理するデータが存在する場合はいつでも実行できるように、高い優先順位を設定できます。優先順位の調節には、注意が必要です。他のスレッドよりも高い優先順位を持つスレッドが実行可能な状態にある場合は、スケジューラはこのスレッドを引き続き実行します。この高い優先順位が変わることなくスレッドもスリープすることがない場合は、それよりも低い優先順位を持つ他のスレッドはまったく実行されません。スレッドの優先順位の調節については、「[srv\\_setpri](#)」(460 ページ) を参照してください。

Open Server が新しいスレッドを確立するときに、そのスレッドが他のスレッドと CPU 時間を完全に共有できるようにするには、その前にスケジューラがある作業を行う必要があります。この新しいスレッドの起動中に、スケジューラは、既存のスレッドが実行できるようにする一連の内部 `srv_yield` 呼び出しを効率的に実行します。その結果、確立済みの実行可能なスレッドが「CPU を占有」して、新しいスレッドの起動を遅らせるように見える場合があります。スレッドが確立されて実行可能になると、そのスレッドの優先順位に応じて CPU 時間を共有します。

実行優先順位が問題となるのは、非プリエンティブ・モードで動作している Open Server アプリケーションの場合のみです。

## ツールと手法

マルチ・スレッド環境でプログラムを作成するには、常にスレッド間の対話に気を配る必要があります。この環境で特に役立つプログラミング・ツールや方法があります。Open Server は、共有リソースへのアクセスを制御するための相互排除セマフォ (ミューテックス)、スレッドが互いに調整し通信することを可能にするメッセージ・キューを提供しています。

## ミューテックス

相互排除セマフォ、つまり「ミューテックス」は、Open Server の論理オブジェクトの 1 つで、最大でも 1 個のスレッドしかこのオブジェクトをロックすることができません。ミューテックスは、共有リソースを保護したり、より複雑なツールを開発したりするのに便利です。

ミューテックスの使用方法を理解するには、次の点に注意してください。

UNIX プラットフォーム上で動作している **Open Server** アプリケーションにおいては、標準入力と標準出力はどのスレッドでも同じです。スレッドが定期的に標準出力に書き込む場合は、アプリケーションのコード上では、標準出力に対して複数のスレッドの出力を指定しないようにしてください。

スレッドの出力が混合することを避ける1つの方法は、`stdout` デバイスにミューテックスを関連付けて、スレッドが `stdout` に書き込む前に必ずミューテックスをロックさせることです。一度に1つのスレッドしかミューテックスをロックできないので、`stdout` に書き込むことができるスレッドは一度に1つだけになります。他のスレッドは、そのミューテックスをロックすることが可能になるまで待機します。

プログラミングの詳細については、[`srv\_createmutex`](#)、[`srv\_lockmutex`](#)、[`srv\_unlockmutex`](#)、[`srv\_deletemutex`](#) のページを参照してください。

## メッセージ・キュー

メッセージ・キューによって、スレッド間の通信が可能になります。メッセージ・キューは、通常、他のスレッドにサービスを行う生成スレッドにデータを送るために使用されます。たとえば、すべてのスレッドは、宛先がログ・ファイルになっているデータを入れるためのメッセージ・キューを作成できます。生成されたスレッドは、キューからメッセージを読み込み、受け取った順にログ・ファイルに書き込むことができます。

メッセージ・キューのメッセージは4バイトの値で、通常は、送受信スレッドが共有するメモリ内にあるデータのアドレスを指すポインタです。キューにメッセージを入れるスレッドと、そのメッセージを読むスレッドのメッセージ・フォーマットは、同じである必要があります。

メッセージが他にデータを参照する場合は、メッセージを送ったスレッドがデータ領域を更新する前に、メッセージを読むスレッドがデータを終了していることを確認する必要があります。メッセージが受け取られる前に、送信ルーチンがメッセージを上書きまたは解放することがないように、メッセージを書き込むルーチンである `srv_putmsgq` には、メッセージがキューから読み込まれるまで送信スレッドをスリープさせるオプションがあります。

プログラミングの詳細については、[`srv\_createmsgq`](#)、[`srv\_putmsgq`](#)、[`srv\_getmsgq`](#)、[`srv\_deletemsgq`](#) のページを参照してください。

## クリティカル・セクションの保護

Open Server がスレッドを中断しないようにするために、`srv_setpri` を呼び出して、一時的にスレッドの優先順位を上げることができます。サーバ・スレッドは、すべて、同一の優先順位で開始されます。この優先順位は、`ospublic.h` に定義されている `SRV_C_DEFAULTPRI` 定数によって表されます。スレッド優先順位の範囲は `SRV_C_LOWPRIORITY` から `SRV_C_MAXPRIORITY` までで、この範囲の中間点は `SRV_C_DEFAULTPRI` です。

Open Server は、必ず、最高優先順位の実行可能なスレッドから再開します。実行可能なスレッドが複数存在し同じ優先順位である場合には、Open Server は最初に実行可能になったスレッドを再開します。1つのスレッドの優先順位を他のスレッドの優先順位よりも高くした場合には、そのスレッドが実行可能でなくなるか、優先順位が下げられるまで、Open Server はそのスレッドを実行し続け、他のスレッドを実行しません。

スレッドの優先順位を上げることは、クリティカル・セクション中に他のスレッドによる介入を防ぐための効果的な手段ですが、その一方で同時実行性に悪影響を及ぼすこともあります。優先順位を上げるということは、1つのスレッドにサーバを独占させることとなります。スレッドの優先順位を `SRV_C_DEFAULTPRI` 以上に上げると、Open Server ランタイム・システムを構成しているスレッドさえも実行できなくなります。このような影響を最小限に抑えるためには、どうしても必要になるまで優先順位の引き上げを遅らせ、できるだけ早くまた下げてください。クリティカル・セクションに不要なコードを入れな

## コールバック・ルーチン

`srv_callback` ルーチンを使って、スレッドのコールバック・ハンドラをインストールできます。Open Server は、スレッドのステータスがユーザの指定したものに変わったときに、ユーザのルーチンを呼び出します。たとえば、あるスレッドが中断したときに実行する `SRV_C_SUSPEND` コールバック・ハンドラをインストールできます。

---

**注意** コールバック・ハンドラをインストールし実行できるかは、プラットフォームに依存します。現在使用しているプラットフォーム上で特定のステータス移行についてコールバック・ハンドラがインストール可能であるかどうかは、`srv_capability` を使って調べてください。

---

表 2-23 は、`srv_callback` によるコールバック・ハンドラのインストールが可能なステータス移行をまとめたものです。

表 2-23 : ステータスの移行

ステータスの移行	意味
SRV_C_EXIT	スレッドは、その実行のために生成されたルーチンの実行を終了したか、接続を切断したクライアントに関連付けられている。ハンドラは、終了するスレッドのコンテキスト内で実行する。
SRV_C_PROCEXEC	Open Server は、レジスタード・プロシージャを実行しようとするときにこのコールバック・ハンドラを呼び出す。ハンドラは、レジスタード・プロシージャを要求したスレッドのコンテキストで実行する。結果として、SRV_C_PROCEXEC コールバック・ハンドラは、クライアントがなんらかのレジスタード・プロシージャ・オペレーションを試行するときに実行する。クライアントがレジスタード・プロシージャを作成、削除、または実行する能力を制限するコールバック・ハンドラをインストールできる。
SRV_C_RESUME	スレッドは再開中である。ハンドラはスケジューラ・スレッドのコンテキストで実行され、スケジューラのスタックを使用する。
SRV_C_SUSPEND	スレッドは中断している。ハンドラは、中断しているスレッドのコンテキストで実行され、そのスタックを使用する。
SRV_C_TIMESLICE	スレッドは、SRV_TIMESLICE、SRV_VIRTCLKRATE、SRV_VIRTTIMER 設定パラメータで指定した時間(タイム・スライス)の間、実行した。このハンドラを使うと、長時間実行しているスレッドに対して、他のスレッドを実行可能にするための <code>srv_yield</code> を呼び出すように信号を送ることができる。

## プログラミングに関する注意事項

Open Server スレッドは、各自のスタックとレジスタ環境を持つ実行のスレッドですが、Open Server ランタイム・システムのオペレーティング・システム・プロセスのリソースも共有しています。

次に、マルチスレッド・プログラミングに関する重要事項を示します。

- グローバル・データ、ファイル・ハンドル、デバイスなどの共有リソースを保護しなければなりません。

共有グローバル・データ項目を更新しているときには、他のスレッドがそのデータをアクセスできないようにする手段をとっていないのであれば、そのスレッドを中断することができるルーチン呼び出さなさい。さもなければ、他のスレッドは矛盾するデータで作業することにもなります。

特定のリソースに対して自身だけがアクセス権を持っているかのように動作するプログラム・ロジックには注意してください。たとえば、グローバル変数の値を使って計算の一部を行い、中断するようなルーチンです。その間に他のスレッドがグローバル変数を変更することが可能であり、矛盾の問題を引き起こします。計算が、完了する前に不正確なものになることがあります。

- 複数のスレッドが実行できるルーチンでは、静的変数は避けてください。

静的変数を変更するルーチンを、複数のスレッドによって呼び出すことが可能な場合は、そのルーチンの複数のインスタンスが競合しないことを保証しなければなりません。そのルーチンが、静的変数のポインタを返すように作られている場合は、スレッドの中断中に変数の内容が変更されることもあるので、データの矛盾がさらに起こりやすくなります。スレッドはそれぞれ専用のスタックを持っているので、自動変数を使用するほうが安全です。アプリケーションはメモリを提供し、そこに結果をコピーしなければなりません。静的変数を使用しなければならない場合は、上記の手段で保護するようにしてください。

- **SRV\_ATTENTION** イベントは、割り込みレベルで実行できます。**SRV\_ATTENTION** ハンドラが操作するアプリケーション構造が、他のイベント・ハンドラやサービス・スレッドなどの非割り込みレベル・コードでも変更またはテストされている場合は、その変更やテストの結果は信頼できません。割り込みレベルの**SRV\_ATTENTION** ハンドラと非割り込みレベル・コードの調整を行うためには、アテンション・レベルの起動とスリープを使用してください。

## 例

サンプル・プログラム *multthrd.c* には、さまざまなマルチスレッド・プログラミングの例が記述されています。

## ネゴシエートされた動作

Open Server アプリケーションは、クライアントとネゴシエートして、さまざまな分野におけるアプリケーションの動作を決定します。ネゴシエーションにはクライアントがログインするときに行われるものもあれば、Open Server ランタイム・システムの存続期間中にアドホックに発生するものもあります。

## ログイン・ネゴシエーション

ログイン時にネゴシエートされる要素はいくつかあります。いくつかは、Open Server が透過的にネゴシエートするものであり、Open Server アプリケーションがアクションを起こす必要はありません。他のものは、アプリケーション呼び出しで明示的に処理されます。ログイン・ネゴシエーションは、常に SRV\_CONNECT イベント・ハンドラ内で起こります。

## 透過的ネゴシエーション

次の要素は、アプリケーションに対して透過的に決定されます。

- 文字データが表示される文字セット。クライアントがログインするときには、さまざまな情報を提供しますが、その中にはロケールに適切な文字セットの名前があります。サーバの文字セットがクライアントの文字セットと異なる場合、Open Server はデータをクライアントの文字セットに変換します。
- 表示する Open Server エラー・メッセージの言語。
- バイト順序：プラットフォームに依存。
- TDS プロトコル・レベル。
- 浮動小数点表現：プラットフォームに依存。

サーバのデフォルトの各国言語と文字セットは、サーバの初期化中に確立されます。

クライアントは、文字セットと各国言語を後で再ネゴシエーションすることができます。「[アドホック・ネゴシエーション](#)」(137 ページ)を参照してください。

## 明示的ネゴシエーション

次の要素は、アプリケーション自体がクライアントとネゴシエートして決定します。

- アプリケーションがデフォルトを使用しない場合には、クライアントのできる要求の種類と、Open Server アプリケーションが返せる応答の種類
- クライアントとサーバが通信するセキュリティ・レベル

クライアントは、ログイン・レコードを送った後に、機能情報を送ります。クライアントと Open Server アプリケーションは、その特定の接続において送信が可能である要求と応答のセットに同意しなければなりません。これらの機能を確立しなければ、さらに要求や応答を送ることはできません。機能の詳細については、「[機能](#)」(25 ページ)を参照してください。

## セキュア接続のネゴシエーション

Open Server アプリケーションは、クライアントとの間に**セキュア接続**を確立することができます。セキュア接続とは、クライアントの ID を詳細に認証し、パスワードを確認した後で確立される接続のことです。

---

**注意** アプリケーションは、アプリケーション自体のセキュリティ・コードを組み込まずに、セキュリティ・サービス・プロバイダが提供する外部セキュリティ・システムを使用できます。サード・パーティのセキュリティ・サービス・プロバイダを使用するように Open Server アプリケーションを設定する方法については、「[セキュリティ・サービス](#)」(196 ページ)を参照してください。

---

アプリケーションは、次の方法のいずれかまたはすべてを使ってこのセキュリティ・チェックを行うことができます。

- クライアントにチャレンジを送り、これに対応する応答をするようにクライアントに要求します。
- クライアントに**暗号化キー**を送ります。クライアントは応答として暗号化パスワードを返し、アプリケーションがこれを複号化して検証します。
- クライアントに**セキュリティ・ラベル**を要求します。クライアントがこれを送り、接続のセキュリティのレベルが確立されます。
- アプリケーション定義のログイン・ハンドシェイクを開始します。



- 透過的なセキュリティ・ハンドシェイクを開始します。このためには、*libtcl.cfg* ファイル内に **security** エントリが指定されていることと、要求されたセキュリティ・サービスのドライバがインストールされている必要があります。「[interfaces ファイルへの変更](#)」(211 ページ) と 「[セキュリティ・サービス](#)」(196 ページ) を参照してください。
- セキュリティ・セッション・コールバックを使って、リモート・サーバとゲートウェイ・クライアントとの間でセキュリティ・セッションのネゴシエーション・データを交換します。「[ダイレクト・セキュリティ・セッションでのフル・パススルー・ゲートウェイ](#)」(219 ページ) および『[Open Client Library/C リファレンス・マニュアル](#)』を参照してください。

アプリケーションがセキュア・ログインをネゴシエートするには、`srv_negotiate` ルーチンを `SRV_CONNECT` イベント・ハンドラの中で実行します。

## アドホック・ネゴシエーション

アプリケーションは、サーバが稼働中であればいつでも、いくつかの要素をクライアントとネゴシエートまたは再ネゴシエートできます。アドホック・ネゴシエーションは、`SRV_LANGUAGE` イベント・ハンドラか `SRV_OPTION` ハンドラ内で起こります。クライアントは、次のことができます。

- `Transact-SQL` 言語コマンドまたはオプション・コマンドを使用して、文字セットと各国言語の再ネゴシエーションを行います。
- `Transact-SQL` 言語コマンドまたはオプション・コマンドを使用して、クエリ処理動作の特性を決定します。クライアントは、特定のオプションの現在のステータスを要求できるだけではなく、オプションの設定やクリアも要求することもできます。

`SRV_OPTION` イベントの説明とオプションのリストについては、「[オプション](#)」(138 ページ) を参照してください。

各国言語と文字セットのネゴシエーションの詳細については、「[国際化サポート](#)」(111 ページ) で説明しています。

『[ASE リファレンス・マニュアル](#)』および「[セキュリティ・サービス](#)」(196 ページ) を参照してください。

### 例

サンプル・プログラム *ctos.c* には、ネゴシエーション・ログインのコードが記述されています。

## オプション

Adaptive Server Enterprise では、クエリ処理をどのように扱うかをクライアントが決めることができます。クエリ処理の動作に関するさまざまな設定可能オプションがあります。詳細については、『ASE リファレンス・マニュアル』の「set コマンド」を参照してください。

Open Server アプリケーションは、クエリ処理オプションに関するクライアント要求に応答することができます。

クライアント・アプリケーションは、次の2つの方法のどちらかで、Adaptive Server Enterprise のクエリ処理オプションの現在の値を設定、クリア、要求できます。

- Transact-SQL 言語コマンドを使用する
- オプション・コマンドを発行する

アプリケーションは、クライアントが要求するオプションのために必要になる言語コマンドを発行すると予期した場合には、SRV\_LANGUAGE イベント・ハンドラでそのような要求を解析するためのコードを含んでいなければなりません。

クライアント・オプション・コマンドは、SRV\_OPTION イベントをトリガします。アプリケーションは、このような要求に対して `srv_options` コマンドを使って、SRV\_OPTION イベント・ハンドラから応答します。

## SRV\_OPTION イベント・ハンドラの内部構造

クライアントは、オプションの設定やクリア、またはその現在の値を返すように要求できます。これらのコマンドはいずれも、SRV\_OPTION イベントをトリガします。SRV\_OPTION イベント・ハンドラを使用して、アプリケーションは次のことを行います。

- 1 `cmd` 引数を `CS_GET` に設定して `srv_options` を呼び出します。クライアントが発行したコマンドの種類 (`SRV_SETOPTION`、`SRV_CLEAROPTION`、または `SRV_GETOPTION`) が、`optcmdp` に返されます。オプション自体は、`optionp` に返されます。`*bufp` は、そのオプションに関連するすべての有効値を含みます。  
  
たとえば、クエリの影響を受けるローの数を Adaptive Server Enterprise がレポートしないようにクライアントが要求した場合は、`optcmdp` に `SRV_SETOPTION` が格納され、`*optionp` に `CS_OPT_NOCOUNT` が格納され、`*bufp` に `CS_TRUE` が格納されます。
- 2 `optcmdp` が `SRV_SETOPTION` または `SRV_CLEAROPTION` の場合、スタンドアロンの Open Server アプリケーションでは、アプリケーション自身がオプションを設定またはクリアします。アプリケーションがゲートウェイである場合は、リモート・サーバのオプションを操作するための適切なクライアント呼び出しを送信しなければなりません。
- 3 `optcmdp` が `SRV_GETOPTION` の場合は、`cmd` を `CS_SET` に設定し、`optcmd` を `SRV_SENDOPTION` に設定し、`optionp` をクライアントが値を求めているオプションに設定し、`bufp` を現在の値に設定して、アプリケーションで `srv_options` を呼び出します。

## オプションの説明とデフォルト値

表 2-24 に、クライアントが設定、取得、クリアできるオプションと、各オプションのデフォルト値を示します。

表 2-24 : サーバ・オプションの記号定数

記号定数	オプションの役割	デフォルト値
<code>CS_OPT_ANSINULL</code>	このオプションが <code>CS_TRUE</code> に設定されている場合、Adaptive Server Enterprise は " <code>=NULL</code> " と " <code>is NULL</code> " が同一ではないという ANSI 動作を強制する。標準 Transact SQL では、" <code>=NULL</code> " と " <code>is NULL</code> " は同一のものとして扱われる。  このオプションは、" <code>&lt;&gt; NULL</code> " と " <code>is not NULL</code> " の動作にも同様に作用する。	<code>CS_FALSE</code>
<code>CS_OPT_ANSIPERM</code>	このオプションが <code>CS_TRUE</code> に設定されている場合、Adaptive Server Enterprise は <code>update</code> 文および <code>delete</code> 文に対するパーミッションの確認を ANSI に準拠させる。	<code>CS_FALSE</code>

記号定数	オプションの役割	デフォルト値
CS_OPT_ARITHABORT	このオプションが CS_TRUE に設定されている場合、Adaptive Server Enterprise は実行中に演算例外が起こったときにクエリをアボートする。	CS_FALSE
CS_OPT_ARITHIGNORE	このオプションが設定されている場合、クエリ実行中に演算例外が起こったときには、Adaptive Server Enterprise は選択値または更新値を NULL に置き換える。Adaptive Server Enterprise は警告メッセージを返しません。CS_OPT_ARITHABORT または CS_OPT_ARITHIGNORE のどちらも設定されていない場合は、Adaptive Server Enterprise は NULL 値による置き換えを行い、クエリの実行完了後に警告メッセージを出力します。	CS_FALSE
CS_OPT_AUTHOFF	現在のサーバ・セッションについて、指定された権限レベルを無効にする。ユーザのログイン時に、そのユーザに与えられた権限がすべて自動的に有効になる。	適用しない
CS_OPT_AUTHON	現在のサーバ・セッションについて、指定された権限レベルを有効にする。ユーザのログイン時に、そのユーザに与えられた権限がすべて自動的に有効になる。	適用しない
CS_OPT_CHAINXACTS	このオプションが CS_TRUE に設定されている場合、Adaptive Server Enterprise は連鎖トランザクション動作を使用する。  連鎖トランザクション動作は、各サーバ・コマンドが別個のトランザクションとみなされていることを意味する。  非連鎖トランザクション動作は、トランザクションを定義する明示的な <code>commit transaction</code> 文を必要とする。	CS_FALSE (非連鎖トランザクション動作)
CS_OPT_CURCLOSEONXACT	このオプションが CS_TRUE に設定されている場合、トランザクション領域内でオープンされたカーソルはすべて、トランザクションが完了するとクローズされる。	CS_FALSE
CS_OPT_CURREAD	現在の読み込みレベルを指定する、セキュリティ・ラベルを設定する。	NULL
CS_OPT_CURWRITE	現在の書き込みレベルを指定する、セキュリティ・ラベルを設定する。	NULL
CS_OPT_DATEFIRST	このオプションは、週の「最初」の日と見なす曜日を設定する。	us_english の場合、デフォルトは CS_OPT_SUNDAY

記号定数	オプションの役割	デフォルト値
CS_OPT_DATEFORMAT	このオプションは、 <code>datetime</code> または <code>smalldatetime</code> データ入力のための日付の要素 (月/日/年) の順番を設定する。	<code>us_english</code> の場合、デフォルトは <code>CS_OPT_FMTMDY</code>
CS_OPT_FIPSFLAG	このオプションが <code>CS_TRUE</code> に設定されている場合、Adaptive Server Enterprise は送られる非標準 SQL コマンドすべてにフラグを立てる。	<code>CS_FALSE</code>
CS_OPT_FORCEPLAN	このオプションが <code>CS_TRUE</code> に設定されている場合、Adaptive Server Enterprise はクエリの "from" 句にリストされている順序でテーブルをジョインする。	<code>CS_FALSE</code>
CS_OPT_FORMATONLY	このオプションが <code>CS_TRUE</code> に設定されている場合、Adaptive Server Enterprise は <code>select</code> クエリに対して、データ自体ではなくデータの説明を返す。	<code>CS_FALSE</code>
CS_OPT_GETDATA	このオプションが <code>CS_TRUE</code> に設定されている場合は、Adaptive Server Enterprise から <code>insert</code> 、 <code>delete</code> 、 <code>update</code> の各コマンドに関する情報が返される。この情報は、メッセージ結果セットとパラメータの形で返される。アプリケーションは、これを利用すると、挿入または削除されるローが格納されるテンポラリ・テーブルの名前を構築できる。更新は、挿入と削除から成ることに注意。	<code>CS_FALSE</code>
CS_OPT_IDENTITYOFF	テーブルの <code>identity</code> カラムへの挿入を実行できないようにする。Adaptive Server Enterprise のマニュアルで <code>set</code> コマンドを参照してください。	適用しない
CS_OPT_IDENTITYON	テーブルの <code>identity</code> カラムへの挿入を有効にする。Adaptive Server Enterprise のマニュアルで <code>set</code> コマンドを参照してください。	適用しない
CS_OPT_ISOLATION	このオプションは、トランザクションの分離レベルを指定するために使う。有効値は、 <code>CS_OPT_LEVEL1</code> と <code>CS_OPT_LEVEL3</code> 。 <code>CS_OPT_ISOLATION</code> を <code>CS_OPT_LEVEL3</code> に設定すると、トランザクション内の <code>select</code> クエリで指定されたテーブルの全ページを、トランザクションの全期間にわたってロックさせる。	<code>CS_OPT_LEVEL1</code>
CS_OPT_NOCOUNT	このオプションを指定すると、各 SQL 文の影響を受けるローの数に関する情報が Adaptive Server Enterprise から返されなくなる。	<code>CS_FALSE</code>
CS_OPT_NOEXEC	このオプションが <code>CS_TRUE</code> に設定されている場合、Adaptive Server Enterprise はクエリをコンパイルはするが、実行はしない。このオプションは、 <code>CS_OPT_SHOWPLAN</code> とともに使われる。	<code>CS_FALSE</code>

記号定数	オプションの役割	デフォルト値
CS_OPT_PARSEONLY	このオプションが設定されている場合、サーバはクエリの構文をチェックし、必要に応じてエラー・メッセージを返すが、クエリを実行することはない。	CS_FALSE
CS_OPT_QUOTED_IDENT	このオプションが CS_TRUE に設定されている場合、Adaptive Server Enterprise は二重引用符内のすべての文字列を識別子として扱う。	CS_FALSE
CS_OPT_RESTREES	このオプションが設定されていると、Adaptive Server Enterprise はクエリの構文はチェックしても実行はせず、必要に応じてクライアントに対して解析解決ツリー(通常のロー結果セットの image カラムの形式)やエラー・メッセージを返す。	CS_FALSE
CS_OPT_ROWCOUNT	このオプションが設定されている場合、select 文実行時に Adaptive Server Enterprise から返される通常ローの数は、指定された最大数までとなる。このオプションは、返される計算ローの数を制限しない。  CS_OPT_ROWCOUNT の動作は、他のオプションとは多少異なる。DBROWCOUNT は常にオンに設定され、オフになることはありません。CS_OPT_ROWCOUNT を 0 に設定するとデフォルトの動作となり、select 文によって生成されたすべてのローを返す。したがって、CS_OPT_ROWCOUNT をオフにするには、カウント 0 で設定をオンにする。	0 (すべてのローが返される)
CS_OPT_SHOWPLAN	このオプションが CS_TRUE に設定されている場合には、Adaptive Server Enterprise はコンパイル終了後に処理プランの記述を作成し、クエリの実行を継続する。	CS_FALSE
CS_OPT_STATS_IO	このオプションは、Adaptive Server Enterprise の内部 I/O 統計を各クエリ後にクライアントに返すかどうかを決定する。	CS_FALSE
CS_OPT_STATS_TIME	このオプションは、Adaptive Server Enterprise の解析、コンパイル、および実行時間統計を各クエリ後にクライアントに返すかどうかを決定する。	CS_FALSE
CS_OPT_STR_RTRUNC	このオプションが CS_TRUE に設定されている場合、Adaptive Server Enterprise は、文字データの右側のトランケーションについて ANSI 標準に準拠する。	CS_FALSE

記号定数	オプションの役割	デフォルト値
CS_OPT_TEXTSIZE	このオプションは、Adaptive Server Enterprise のグローバル変数 @@textsize の値を変更する。この変数は、Adaptive Server Enterprise が返す text 値または image 値のサイズを制限する。このオプションを設定するときは、Adaptive Server Enterprise が返す必要がある最も長い text 値または image 値の長さをバイト単位で示すパラメータを使用すること。	32,768 バイト
CS_OPT_TRUNCIGNORE	このオプションが CS_TRUE に設定されている場合には、Adaptive Server Enterprise はトランケート・エラーを無視する。これが標準 ANSI 動作。  このオプションが CS_FALSE に設定されている場合、変換の結果トランケートが生じたときには Adaptive Server Enterprise はエラーを出す。	CS_FALSE

「[srv\\_options](#)」(365 ページ) に、各オプションの有効値とデータ型を示します。

## 例

サンプル・プログラム *ctos.c* には、クライアント・オプション・コマンドを処理するコードが記述されています。

## パラメータ

### バッチ・パラメータ

Open Server は、コマンド実行時に複数セットのコマンド・パラメータを受け入れることができます。バッチされたパラメータのサポートを有効にするには、SRV\_S\_PARAM\_BATCHING プロパティを CS\_TRUE に設定します。

これで、コマンドに複数セットのコマンド・パラメータが含まれている場合に `srv_xferdata()` に 2 つのリターン・コードが新しく割当てられるようになります。

- **CS\_PARAMS\_MORE** - パラメータが正常にコピーされ、複数のパラメータがバッチ内に存在することを示します。
- **CS\_PARAMS\_END** - パラメータが正常にコピーされたことを示します。これがバッチ内の最後のパラメータ・セットです。

## 部分更新

Open Client および Open Server は、text カラムと image カラムの部分更新をサポートしています。部分更新では、置換、削除、または挿入する text フィールドまたは image フィールドの部分を指定できます。また、フィールド全体を修正するのではなく、その部分だけを更新することもできます。『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

---

**注意** 現在、Adaptive Server Enterprise は、text データと image カラムの部分更新をサポートしていません。

---

## Open Server の設定

この項では、部分更新をサポートするために Open Server を設定する方法について説明します。

### *sp\_mda*

*sp\_mda* は、サーバからメタデータを取得するストアド・プロシージャです。部分更新をサポートするには、Open Server アプリケーションが *sp\_mda* ストアド・プロシージャを定義し、Open Client アプリケーションが使用する必要がある *updatetext* 構文を指定する必要があります。

Open Client アプリケーションは、次のパラメータと値を使用して *sp\_mda* を呼び出す必要があります。

パラメータ	値	説明
<i>clnttype</i>	5	5 は、クライアントが Client-Library であることを示しています。
<i>mdaversion</i>	1	



パラメータ	値	説明
clientversion	0	clientversion は、クライアントのバージョンを示すオプションのパラメータです。デフォルトは0です。

サーバが部分更新をサポートする場合、sp\_mda は次の値を返します。

パラメータ	値
mdinfo	“UPDATETEXT”
querytype	2
query	<i>updatetext_syntax</i> 例： updatetext ?? ?{NULL   ?}{NULL   ?} ここで、“?” は updatetext パラメータを示しています。

『Mainframe Connect DB2 UDB Options for IBM CICS and IMS Installation and Administration Guide』を参照してください。sp\_mda のサンプル実装については、`$$SYBASE/$$SYBASE_OCS/sample/srvlibrary/updtext.c` を参照してください。

## SRV\_T\_BULKTYPE

クライアントにより送信された部分更新済みデータを正しく取得するには、Open Server アプリケーションは SRV\_T\_BULKTYPE を SRV\_TEXTLOAD、SRV\_UNITEXTLOAD、SRV\_IMAGELOAD に設定する必要があります。「SRV\_T\_BULKTYPE」(177 ページ) を参照してください。

## ハンドラ

SRV\_LANGUAGE および SRV\_BULK ハンドラは、Open Server にインストールする必要があります。Open Server は、SRV\_LANGUAGE を使用して、Client-Library から updatetext 文を取得します。一方、SRV\_BULK は、ct\_send\_data() を通して送信されたデータを取得します。

『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

## パススルー・モード

Open Client アプリケーションと Adaptive Server Enterprise 間でゲートウェイとしての役割を果たす Open Server アプリケーションは、内容を確かめずにクライアントとサーバの間で TDS パケットのやり取りを行うことができます。TDS パケットをこのように処理する Open Server は、パススルー・モードで動作していると言います。

Open Server ゲートウェイ・アプリケーションがクライアントから受け取った TDS 情報をアンパックしたり、Adaptive Server Enterprise に送信する前に再パックしたりする必要がないので、パススルー・モードは非常に効率的です。

Open Client Server 12.5.1 以前のパススルー・モードでは、クライアントから要求されるパケット・サイズを Open Server によりサポートされる最大サイズに制限することにより、ネゴシエート済みパケット・サイズの妥当性を保証しています。

サーバ指定のパケットサイズをサポートするリモート・サーバが Open Server 内での設定より長いパケットサイズを指定する場合、長い方のパケットサイズが `SRV_S_NETBUFSIZE` の設定にかかわらず使用されます。

パススルー・モードには、次の 2 種類があります。

- 通常パススルー・モード
- イベント・ハンドラ・パススルー・モード

どちらの種類のパススルー・モードも、パススルー・ルーチン

`srv_recvpassthru`、`ct_sendpassthru`、`ct_recvpassthru`、`srv_sendpassthru` を使用します。相違点は次のとおりです。

- 通常パススルー・モードでは、Open Server アプリケーションはイベントを認識し、イベント・ハンドラをトリガします。これらのイベント・ハンドラは、パススルー・ルーチンを呼び出すようにコーディングされます。

「[通常パススルー・モード](#)」(147 ページ) を参照してください。

- イベント・ハンドラ・パススルー・モードでは、Open Server アプリケーションは接続されているイベントのほとんどの種類を認識しません。代わりに、接続のネットワーク読み込みが完了するたびに、フル・パススルー・イベント・ハンドラがトリガされます。フル・パススルー・イベント・ハンドラは、パススルー・ルーチンを呼び出すようにコーディングされます。

「イベント・ハンドラ・パススルー・モード」(149 ページ)を参照してください。

DB-Library にも、パススルー・モードをサポートするルーチンがあります。詳細については、『Open Client DB-Library/C リファレンス・マニュアル』を参照してください。

## 通常パススルー・モード

Sybase では、最初はこの種類のパススルー・モードだけをサポートしていました。

通常のパススルー・モードでは、Open Server はイベント (SRV\_LANGUAGE、SRV\_RPC など) を認識し、適切なイベント・ハンドラをトリガします。各イベント・ハンドラは、パススルー・ルーチン呼び出すようにコーディングする必要があります。

## パススルー・モードでの TDS プロトコル・レベルのネゴシエーション

Sybase クライアントとサーバが接続するときには、互いに使用する TDS プロトコルをまず合わせますが、これは通常、両方のプログラムが認識できる最新バージョンのプロトコルです。「ネゴシエートされた動作」(135 ページ)を参照してください。

Open Server ゲートウェイ・アプリケーションがパススルー・モードで動作している場合、TDS パケットの作成と解釈は、ゲートウェイではなく、リモート Sybase クライアントと Adaptive Server Enterprise によって行われます。したがって、TDS ネゴシエーションは2つのリモート・プログラム間で行われることとなります。ゲートウェイは、両者間の応答を伝達することによって、このネゴシエーションを簡単にしなければなりません。TDS ネゴシエーション・プロセスは SRV\_CONNECT イベント・ハンドラ内部で起こらなければならない、次の手順が必要です。

- 1 次のいずれかのプロパティを設定します。
  - スレッドが通常パススルー・モードを使用することを示す SRV\_T\_PASSTHRU
  - スレッドがイベント・ハンドラ・パススルー・モードを使用することを示す SRV\_T\_FULLPASSTHRU

srv\_getloginfo および ct\_setloginfo がクライアント/サーバ機能をパススルー・モードに正しくネゴシエートするために、これらのプロパティのどちらかを設定してください。

- 2 srv\_getloginfo - CS\_LOGININFO 構造体を割り付け、クライアント・スレッドからのログイン情報を格納します。
- 3 ct\_setloginfo - 手順 2 で取得したログイン情報を使用して CS\_LOGININFO 構造体を準備します。
- 4 クライアント・アプリケーションがネットワークベースの認証を使用している場合は、次の手順に従ってクライアントのセキュリティ・プリンシパル名を転送します。セキュリティ・プリンシパル名は CS\_LOGININFO 構造体の一部ではないので、これらの手順が必要です。
  - srv\_thread\_props(..CS\_GET, SRV\_T\_USER) を呼び出して、クライアントのセキュリティ・プリンシパル名を取得します。
  - ct\_con\_props(..CS\_SET, CS\_USERNAME) を呼び出して、ターゲット・サーバへの接続のプリンシパル名を設定します。
- 5 ct\_connect を呼び出して、リモート・サーバにログインします。
- 6 ct\_getloginfo - CS\_CONNECTION 構造体から、新しく割り付けられた CS\_LOGININFO 構造体に、ログイン応答情報を転送します。
- 7 srv\_setloginfo - 手順 6 で取得したリモート・サーバの応答をクライアントに送信し、CS\_LOGININFO 構造体を解放します。

## 通常パススルー・モードの使用

通常の TDS パススルーは、SRV\_ATTENTION、SRV\_CONNECT、SRV\_DISCONNECT、SRV\_START、SRV\_STOP 以外のどのイベント・ハンドラでも起こります。

クライアント要求は、1つまたは複数の TDS パケットのストリームの形で届きます。info 引数が `SRV_I_PASSTHRU_MORE` に設定されている間は、ハンドラは繰り返し `srv_recvpassthru` を呼び出します。各パケットを受け取るたびに、ハンドラはそのパケットをリモート Adaptive Server Enterprise または Open Server に渡すために `ct_sendpassthru` を呼び出します。リモート・サーバは、接続クライアントから直接受け取るのとまったく同じ TDS ストリームを受け取ります。

---

**警告！** TDS の最新バージョンでは、1つのバッチで複数のコマンドを送る機能が導入されています。最初のコマンドだけがイベント・ハンドラをトリガします。残りのコマンドについては、Open Server はイベント・ハンドラを呼び出しません。

---

Client-Library ルーチンである `ct_recvpassthru` は、接続時に到着する TDS パケットを受け取ります。`srv_sendpassthru` Server-Library ルーチンは、パケットをクライアントにそのまま送ります。`CS_PASSTHRU_MORE` が返される間は、`ct_recvpassthru` ルーチンは別の TDS パケットを受け取ります。

## 例

サンプル・プログラム `fullpass.c` には、パススルー・モード・ゲートウェイの例が記述されています。

## イベント・ハンドラ・パススルー・モード

この種類のパススルー・モードでは、Open Server はほとんどのタイプのイベントを認識しません。その代わりに Open Server は、接続からのネットワーク読み込みが完了するたびに、フル・パススルー・イベント・ハンドラを起動します。

イベント・ハンドラ・パススルー・モードでは、パケットごとのセキュリティ・サービス (暗号化など) を使用するクライアント/サーバ接続がパススルー・モードを使用できるように設計されています。

通常パススルー・モードでは、Open Server がパケットを解釈して特定のイベントを識別する必要があります。パケットが暗号化されている場合は、これは不可能です。

スレッドに対してイベント・ハンドラ・パススルー・モードを使用するには、次の手順に従います。

- フル・パススルー・イベント・ハンドラをコーディングしてインストールします。「フル・パススルー・イベント・ハンドラのコーディングとインストール」(150 ページ)を参照してください。
- 特定のスレッドに対してイベント・ハンドラ・パススルー・モードを有効にするために、Open Server 接続ハンドラ内で `SRV_T_FULLPASSTHRU` を `CS_TRUE` に設定します。「スレッドに対するイベント・ハンドラ・パススルー・モードの有効化」(151 ページ)を参照してください。
- クライアントとターゲット・サーバとの間で TDS プロトコル・レベルをネゴシエートするルーチンを呼び出します。「TDS プロトコル・レベルのネゴシエーション」(151 ページ)を参照してください。

## フル・パススルー・イベント・ハンドラのコーディングとインストール

フル・パススルー・イベント・ハンドラのプロトタイプは次のとおりです。

```
CS_RETCODE CS_PUBLIC func (SRV_PROC *sproc);
```

フル・パススルー・イベント・ハンドラは、次のルーチンを呼び出してパケットの送受信を行います。

- `srv_recvpasssthru`
- `ct_sendpasssthru`
- `ct_recvpasssthru`
- `srv_sendpasssthru`

`srv_recvpasssthru/ct_sendpasssthru` ループの実行中はアテンション・イベントを転送できません。イベント・ハンドラのコードとアテンション・ハンドラのコードにロジックを追加して、コマンド全体がリモート・サーバに転送されるまでアテンション・イベントが転送されないようにしてください。

フル・パススルー・イベント・ハンドラは `CS_SUCCEED` を返して、正常終了をレポートする必要があります。戻り値が `CS_SUCCEED` 以外の場合は、現在の Open Server スレッドが強制終了されます。

フル・パススルー・イベント・ハンドラをインストールするには、`srv_handle` の `event` パラメータを `SRV_FULLPASSTHRU` に、`handler` パラメータをハンドラ・ルーチンのアドレスに設定して `srv_handle` を呼び出してください。

## スレッドに対するイベント・ハンドラ・パススルー・モードの有効化

特定のスレッドに対してイベント・ハンドラ・パススルー・モードを有効にするには、Open Server 接続ハンドラ内で `SRV_T_FULLPASSTHRU` スレッド・プロパティを `CS_TRUE` に設定します。

イベント・ハンドラ・パススルー・モードが有効になると、接続からのネットワーク読み込みが完了するたびに、Open Server はフル・パススルー・ハンドラを起動します。

このスレッドでは `SRV_LANGUAGE`、`SRV_RPC`、`SRV_BULK`、`SRV_CURSOR`、`SRV_MSG`、`SRV_OPTION`、または `SRV_DYNAMIC` タイプのイベントは発生しません。

ただし、`SRV_ATTENTION` イベントは発生します。Open Server アプリケーションは `SRV_ATTENTION` ハンドラをインストールして、キャンセル要求を正しく処理する必要があります。

## TDS プロトコル・レベルのネゴシエーション

イベント・ハンドラ・パススルー・モードを使用するゲートウェイ・アプリケーションでは、通常パススルー・モードを使用するアプリケーションとまったく同じ方法で、クライアント・アプリケーションとターゲット・サーバとの間の TDS プロトコル・レベルのネゴシエーションが行われます。

アプリケーションの接続ハンドラ内で、`SRV_FULLPASSTHRU` を `CS_TRUE` に設定してから、`srv_getloginfo`、`ct_setloginfo`、`ct_getloginfo`、`srv_setloginfo` の各ルーチン呼び出ししてください。

「パススルー・モードでの TDS プロトコル・レベルのネゴシエーション」(147 ページ) を参照してください。

## パラメータとロー・データの処理

### 用語についての注意

「パラメータ・データ」という用語は、クライアントから取得した、またはクライアントに返されたパラメータを指します。その中には入力パラメータもあれば、出力パラメータ、つまりリターン・パラメータもあります。リターン・パラメータは2段階に分けて処理されます。つまり、Open Server アプリケーションによってプログラム変数に読み込まれた時点で部分的に処理され、クライアントに返されるときに処理が完了します。

### Open Server のデータ処理モデル

Open Server では、3つのルーチンが共同作業としてクライアントからパラメータ・データやフォーマットを取得し、ロー・データを送り、またクライアントにパラメータとそのフォーマットを返します。これらのルーチンとは、`srv_descfmt`、`srv_bind`、`srv_xferdata` です。

アプリケーションは、パラメータを提供したり結果を要求するようなクライアント・コマンドはすべて、これらのルーチンを使って処理します。RPC コマンド、言語コマンド、カーソル・コマンド、動的 SQL コマンド、メッセージ・コマンド、ネゴシエートされたログイン・コマンドはすべて、これに当てはまります。

これらの3つのルーチンはそれぞれ `type` 引数を持ち、ここに記述、バインド、または転送されるデータの型を示します。たとえば、カーソル・コマンドの入力パラメータのフォーマットを記述するときには、`type` は `SRV_CURDATA` に設定されますが、結果ローを処理するときには、`type` は `SRV_ROWDATA` に設定されます。`type` の有効値のリストについては、第3章の各ルーチンのページを参照してください。

また、このルーチンは3つとも、`cmd` 引数を持ちます。これは、データ・フローの方向を示します。`CS_GET` の値は Open Server アプリケーションにクライアントから情報を取得することを指示しますが、`CS_SET` の値は、アプリケーションがクライアントに結果を返すことを指示します。

アプリケーションは、これらのルーチンを使用して、次の処理を実行できます。



- `SRV_RPC`、`SRV_CURSOR`、`SRV_DYNAMIC`、`SRV_MSG`、または `SRV_CONNECT` イベント・ハンドラ内で、入力およびリターン・パラメータ情報を取得します。
- `SRV_RPC`、`SRV_CURSOR`、`SRV_DYNAMIC`、`SRV_LANGUAGE`、または `SRV_MSG` イベント・ハンドラ内で結果ロー情報を返信します。
- `SRV_LANGUAGE` または `SRV_RPC` ハンドラ内でリターン・パラメータ情報を返信します。

## パラメータの取得

パラメータを処理するには、アプリケーションは次の手順に従います。

- 1 コマンドに含まれているパラメータ数 (パラメータがある場合) を判断するために、`srv_numparams` を呼び出します。
- 2 `srv_descfmt` を呼び出して、各パラメータの記述を取得します。この記述の中に、パラメータがリターン・パラメータであるかどうかの情報も含まれています。リターン・パラメータがある場合には、取得処理はその時点で終わります。パラメータが入力パラメータである場合には、アプリケーションは手順3と4を続けて行います。
- 3 ネットワーク経由でクライアントから入ってくるパラメータ・データを保存するプログラム変数を提供するために、`srv_bind` を呼び出します。
- 4 手順3で指定されたアプリケーション・プログラム変数にクライアント・データを転送するために、`srv_xferdata` を呼び出します。

リターン・パラメータは、クライアントから取得したときには有効なデータを含んでいません。アプリケーションがリターン・パラメータをクライアントに返すときに、有効データを入力します。**Open Server** は、リターン・パラメータ・フォーマットを、プログラム変数フォーマットからクライアント・フォーマットに透過的に変換します。

`SRV_LANGUAGE` ハンドラ内では、最初に実際のパラメータを取得しなくても、アプリケーションは無区別の言語ストリームからリターン・パラメータを「構成」できます。詳細については、「[言語データ・ストリームでのパラメータの返送](#)」(157 ページ)を参照してください。

`srv_xferdata` は全パラメータ・ストリームのために一度だけ呼び出されるのに対し、`srv_descfmt` と `srv_bind` は各パラメータごとに呼び出されます。アプリケーションは、すべてのパラメータが記述されバインドされるまでは `srv_xferdata` を呼び出してはいけません。

アプリケーションがクライアントから情報を取得するので、アプリケーションはこの3つのルーチンを、それぞれの `cmd` 引数を `CS_GET` に設定して呼び出さなければなりません。

### ローの返送

ロー・データの処理には、次の3つの基本手順が必要です。

- 1 `srv_descfmt` を呼び出し、ロー内の各カラムを記述します。
- 2 `srv_bind` を呼び出し、アプリケーションがロー・データを保存した場所を示し、そのフォーマットを識別します。
- 3 `srv_xferdata` を呼び出して、手順2で指定したアプリケーション・プログラム変数からクライアントにデータを転送します。

`srv_descfmt` ルーチンはローのカラムごとに一度ずつ呼び出さなければなりません。が、`srv_xferdata` ルーチンと `srv_bind` ルーチンは結果ローの数だけ呼び出します。すべてのカラムが記述されバインドされるまでは、アプリケーションは `srv_xferdata` を呼び出せません。

アプリケーションがクライアントに情報を返すので、アプリケーションはこの3つのルーチンを、それぞれの `cmd` 引数を `CS_SET` に設定して呼び出します。

### リターン・パラメータの返送

リターン・パラメータの処理には、次の2つの基本手順が必要です。

- 1 `srv_bind` を呼び出し、アプリケーションがリターン・パラメータ・データを保存した場所を示し、そのフォーマットを識別します。
- 2 `srv_xferdata` を呼び出して、手順2で指定したアプリケーション・プログラム変数からクライアントにリターン・パラメータ・データを転送します。

アプリケーションがクライアントに情報を返すので、アプリケーションはこの2つのルーチンをそれぞれの `cmd` 引数を `CS_SET` に設定して呼び出します。

リターン・パラメータがテキスト・ストリームから「構成」されている場合、バインドと転送だけでなく、記述する必要もあります。詳細については、「[言語データ・ストリームでのパラメータの返送](#)」(157 ページ)を参照してください。

## 記述、バインド、転送

この項では、記述、バインド、および転送処理の詳細について説明します。

### 記述

`srv_descfmt` ルーチンは、アプリケーションがクライアントの予期しているフォーマットでデータを返すために必要な情報を **Open Server** アプリケーションに提供します。概念的に、このルーチンは、クライアントがどのようにデータを見たか(`CS_GET`)または見るのか(`CS_SET`)に関する情報を伝達します。`srv_descfmt` ルーチンは、さまざまなパラメータとローのプロパティを取得したり設定したりします。

これらのプロパティには、次のような情報が含まれます。

- パラメータまたはカラムの名前
- パラメータまたはカラムの名前の長さ
- パラメータまたはカラムの番号 (ストリームの先頭のパラメータまたはカラムの番号は 1 から開始される)
- パラメータまたはカラムのデータ型
- パラメータまたはカラムで `null` を設定できるかどうか
- パラメータがリターン・パラメータであるかどうか

`srv_descfmt` への `clfmt` 引数は、この情報を含んでいる `CS_DATAFMT` 構造体を指します。詳細については、「[CS\\_DATAFMT 構造体](#)」(57 ページ)を参照してください。

### バインド

**Open Server** アプリケーションは、クライアントから受け取るデータを調べるために、そのデータをローカル・プログラム変数の形で保存しなければなりません。アプリケーションは、`srv_bind` を呼び出すときに、パラメータまたはカラムのデータをローカル・プログラム変数と関連付け、その変数のフォーマットを記述します。

cmd を CS\_GET に設定して srv\_bind を呼び出すことによって、クライアントから送られるデータの保管位置を Open Server に指示します。cmd を CS\_SET に設定して srv\_bind を呼び出すことによって、クライアントに返すデータがどこにあるかを Open Server に指示します。

srv\_bind への osfmt 引数は、ローカル・プログラム変数に関する構成情報を含んでいる CS\_DATAFMT 構造体を指します。

## 転送

srv\_xferdata ルーチンは、srv\_bind 呼び出しで指定されたローカル・プログラム変数にデータを出し入れします。cmd が CS\_GET に設定されているときには、srv\_xferdata はクライアントからの入力パラメータ・データを変数に入れます。cmd が CS\_SET に設定されているときには、このルーチンはカラムとリターン・パラメータ・データをローカル・プログラム変数から取り出しクライアントに送ります。

---

**注意** 現時点では、srv\_senddone を実行するとフォーマットとカラム情報がネットワークにフラッシュされますが、今後のバージョンでは、フラッシュされなくなります。アプリケーションは、必ず srv\_xferdata を使ってネットワークに情報をフラッシュしてください。

---

srv\_bind、srv\_descfmt、srv\_xferdata の詳細については、それぞれのページを参照してください。

## 自動変換

アプリケーションがデータを取得するときに、アプリケーションのローカル・プログラム変数のフォーマットと異なるフォーマットでクライアントがデータを送信した場合、Open Server はデータをローカル・フォーマットに変換します。アプリケーションがクライアントにデータを返送するときに同じ状況が起きた場合には、Open Server はデータをクライアント・フォーマットに変換します。

## 言語データ・ストリームでのパラメータの返送

言語データ・ストリームには、パラメータの概念はありません。ただし、テキスト・ストリームを解析できるように装備されている **Open Server** アプリケーションは、受信ストリームからリターン・パラメータを「構成」できます。その後、パラメータにデータをロードして、記述/バインド/転送のプロシージャを使用して返送できます。

たとえば、クライアントがリターン・パラメータを含む **Transact-SQL** スタアド・プロシージャを送るとします。このクエリを予期している **Open Server** アプリケーションは、"output = @var" という文字列 (*var* はリターン・パラメータのプレースホルダ) を解析し、*var* のフォーマット情報とデータを返すことができます。

アプリケーションが *cmd* を **CS\_SET** に、*type* を **SRV\_RPCDATA** に設定して *srv\_descfmt* を呼び出すことができるのは、言語イベント・ハンドラからだけです。

## 例

サンプル・プログラム *ctos.c* は、一連の記述/バインド/転送の呼び出しを使ってパラメータとカラムのデータを処理します。

## プロパティ

プロパティは、**Open Server** アプリケーションの動作のさまざまな内容を定義します。**Open Server** プロパティは次の3つに分類されます。

- コンテキスト・プロパティ
- サーバ・プロパティ
- スレッド・プロパティ

コンテキスト・プロパティとサーバ・プロパティは、全体として **Open Server** アプリケーションに付属しています。これらのプロパティはサーバワイドな動作を制御して、すべてのクライアント/サーバ接続に有効です。

スレッド・プロパティは、クライアント・スレッドとサービス・スレッドに付属しています。これらのプロパティのほとんどは取得のみが可能で、設定することはできません。アプリケーションはいくつかのスレッド・プロパティを設定して、接続ごとにいくつかのサーバワイドな属性を上書きできます。

プログラムはプロパティを設定することで、**Open Server** アプリケーションの機能を調節できます。さらにアプリケーションは、情報が必要なときにいくつかのプロパティを取得することもできます。

`cs_config`、`srv_props`、`srv_thread_props` の各プロパティを使用して、コンテキスト・プロパティ、サーバ・プロパティ、スレッド・プロパティをそれぞれ設定します。

「[コンテキスト・プロパティ](#)」(158 ページ)、「[サーバ・プロパティ](#)」(159 ページ)、「[スレッド・プロパティ](#)」(168 ページ)を参照してください。

『[Open Client and Open Server Common Libraries](#) リファレンス・マニュアル』の「`cs_config`」の項と、このマニュアルの [`srv\_props`](#) および [`srv\_thread\_props`](#) のリファレンス・ページを参照してください。

## コンテキスト・プロパティ

コンテキスト・プロパティは、**CS-Library** の `CS_CONTEXT` 構造体内に保管されます。アプリケーションでコンテキスト・プロパティを設定または取得するには、**CS-Library** ルーチン `cs_config` を使用します。このルーチンの詳細については、『[Open Client/Server Common Libraries](#) リファレンス・マニュアル』を参照してください。

コンテキスト・プロパティは3種類あります。

- **CS-Library** 特有のコンテキスト・プロパティ

`cs_config` は、**CS-Library** 固有のコンテキスト・プロパティの値を設定および取得します。`CS_LOC_PROP`を除く `cs_config` によって設定されるプロパティは、**CS-Library** にだけ反映されます。**CS-Library** 特有のコンテキスト・プロパティのリストについては、『[Open Client/Server Common Libraries](#) リファレンス・マニュアル』の「`cs_config`」を参照してください。

- **Client-Library** に固有なコンテキスト・プロパティ

`ct_config` は、Client-Library 固有のコンテキスト・プロパティの値を設定および取得します。`ct_config` によって設定されるプロパティは、Client-Library にだけ反映されます。『Open Client Library/C リファレンス・マニュアル』を参照してください。

- Server-Library 固有のコンテキスト・プロパティ

`srv_props` は、Server-Library 固有のコンテキスト・プロパティの値を設定および取得します。`srv_props` によって設定されるプロパティは、Server-Library にだけ反映されます。

Open Server アプリケーションが設定できるコンテキスト・プロパティは、次のとおりです。

- Open Server が CS-Library エラーを検出した場合に呼び出すルーチン
- Open Server の各国言語、文字セット、ソート順などのローカライゼーション情報
- アプリケーションのデータ領域へのポインタのロケーション。このプロパティを使用すると、アプリケーションは制御情報を Open Server のコンテキストと対応付けることができます。Open Server はこのポインタを使用しません。つまりこのポインタは、Open Server アプリケーション・プログラマの利便性を考慮して提供されています。

これらのコンテキスト・プロパティの設定と取得はどちらも、`cs_config` ルーチンを使用して行うことができます。『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

## サーバ・プロパティ

サーバ・プロパティは `CS_CONTEXT` 構造体内に保管されます。アプリケーションは Server-Library ルーチン `srv_props` を使用して、サーバ・プロパティの設定または取得を行います。

サーバ・プロパティによって、メモリ割り付けルーチンや確立できる物理ネットワーク接続の最大数など、Open Server アプリケーションの動作のさまざまな内容が定義されます。

サーバ・プロパティの内容を有効にするには、アプリケーションが初期化の前にこれらのプロパティを設定する必要があります。初期化の後でサーバ・プロパティが設定された場合は、Open Server がエラーを表示します。

アプリケーションの初期化コードには、次の手順が必要です。

- 1 `cs_ctx_alloc` を呼び出して、`CS_CONTEXT` 構造体を割り付けます。
- 2 `srv_version` ルーチン呼び出して、`Open Server` のバージョン番号を設定します。`srv_version` ルーチンは `CS_CONTEXT` 構造体へのポインタを取ります。
- 3 `srv_props` ルーチン呼び出して、プロパティのデフォルトを設定します。
- 4 `srv_init` ルーチン呼び出して、サーバを初期化します。
- 5 `srv_run` を呼び出して、サーバの稼働を開始します。

設定と取得の両方が可能なプロパティもありますが、いずれか一方しか実行できないプロパティもあります。詳細については、「[srv\\_props](#)」(377 ページ) を参照してください。

表 2-25 :\_サーバ・プロパティ

プロパティ名	定義	注意
<code>SRV_S_ALLOCFUNC</code>	<code>Open Server</code> がメモリを割り付けるのに使用するルーチンのアドレス。	
<code>SRV_S_APICCHK</code>	<code>Server-Library</code> 引数の検証とステータスのチェックを有効 ( <code>CS_TRUE</code> ) にするか無効 ( <code>CS_FALSE</code> ) にするかを示すブール値。	多くの <code>Server-Library</code> ルーチンは内部的に <code>CS-Library</code> ルーチン呼び出す。このため、引数とステータスを完全にチェックするアプリケーション・プログラマは、 <code>cs_config</code> プロパティ <code>CS_NOAPICCHK</code> を <code>CS_FALSE</code> に設定する必要がある。
<code>SRV_S_ATTNREASON</code>	<code>Open Server</code> アプリケーションのアテンション・ハンドラが呼び出された理由。	クライアントのアテンションが <code>SRV_ATTENTION</code> イベントをトリガした場合は <code>SRV_ATTENTION</code> を返す。クライアント接続の切断がそのイベントをトリガした場合は、 <code>SRV_DISCONNECT</code> を返す。
<code>SRV_S_CERT_AUTH</code>	<code>CS_CHAR</code> 信頼された CA 証明書を含むファイルへのパスを指定する。	このプロパティの最大長は <code>SRV_MAXCHAR</code> バイト。
<code>SRV_S_CURTHREAD</code>	アクティブなスレッドの内部制御構造体のアドレス。	スレッド・ライブラリとともに <code>SRV_S_PREEMPT</code> を使用した場合、一部の <code>SRV_S_CURTHREAD</code> 機能が無効になる。



プロパティ名	定義	注意
SRV_S_DEFQUEUESIZE	遅延イベント・キューのサイズ。	
SRV_S_DISCONNECT	このプロパティを <b>CS_TRUE</b> に設定すると、クライアントの接続が切断されたときにアプリケーションの <b>SRV_ATTENTION</b> イベント・ハンドラが呼び出される。	割り込み時にクライアント接続の切断が検出された場合は、割り込みレベルで <b>SRV_ATTENTION</b> イベント・ハンドラを呼び出すことができる。
SRV_S_DSPPROVIDER	ディレクトリ・サービス・プロバイダの名前。デフォルト値はプラットフォームごとに異なる。詳細については、使用しているプラットフォームの『Open Client/Server 設定ガイド』を参照してください。	このプロパティの最大長は <b>SRV_MAXCHAR</b> バイト。
SRV_S_DSREGISTER	<b>Server-Library</b> が起動時に <b>Server-Library</b> 自体をディレクトリ・サービスに登録する必要があることを示すには、 <b>CS_TRUE</b> に設定する。登録を行わないようにするには、 <b>CS_FALSE</b> に設定する。	
SRV_S_ERRHANDLE	<b>Open Server</b> エラー・ハンドラのアドレス。	
SRV_S_FREEFUNC	<b>Open Server</b> がメモリを解放するのに使用するルーチンのアドレス。	
SRV_S_IFILE	<b>Open Server</b> で使用可能な <b>interfaces</b> ファイルの名前。	このプロパティの最大長は <b>SRV_MAXCHAR</b> バイト。
SRV_S_LOGFILE	<b>Open Server</b> が書き込みを行うログ・ファイルの名前。	<b>SRV_S_LOGFILE</b> プロパティは、 <b>srv_init</b> の呼出し後に設定できる。  <b>srv_init</b> を呼び出した後、 <b>bufp</b> を空の文字列 ("") に設定し、 <b>buflen</b> を 0 に設定した状態で <b>SRV_S_LOGFILE</b> プロパティを設定すると、ログ・ファイルが閉じる。  このプロパティの最大長は <b>SRV_MAXCHAR</b> バイト。
SRV_S_LOGSIZE	ログ・ファイルの最大サイズ。ログがこのサイズよりも大きくなった場合、 <b>Open Server</b> はログ・ファイルの現在の内容を <b>currentfilename_old</b> という名前の別のファイルに移動して、現在のログを 0 バイトにトランケートする。	

プロパティ名	定義	注意
SRV_S_MAXLISTENERS	リスナ・スレッドの最大値を制限する。	デフォルト値は <b>CS_MAX_NOMAX</b> 。この値は、新しいリスナ・スレッドを作成するときのみ使用される。このプロパティを <b>SRV_S_NUMLISTENERS</b> の現在の値よりも小さい値に設定すると、リスナは失敗しない。
SRV_S_MSGPOOL	実行時に Open Server アプリケーションに対して使用できるメッセージの数。	Open Server アプリケーションでは、 <b>srv_putmsgq</b> でメッセージを使用する。メッセージは、 <b>srv_getmsgq</b> によって受信されるまで継続して使用される。 <b>SRV_S_MSGPOOL</b> 設定パラメータの値は、アプリケーションでこの2つのルーチンをどのように使用するかに基づいて決定する。
SRV_S_NETBUFSIZE	クライアント接続が使用するネットワーク I/O バッファの最大サイズ。明示的に設定しない場合、 <b>SRV_S_NETBUFSIZE</b> はデフォルトの最大値である 8192 バイトになる。	Open Client Server 12.5.1 以前では、ログイン時にネットワーク・バッファのサイズが決定される。より小さなバッファ・サイズが要求される場合でも、Open Server はメモリ・バッファのサイズを変更しないで、その部分を未使用のままにしておく。したがって、この値を必要以上に大きくしないこと。必要以上に大きくすると、使用されないメモリが割り付けられる。
SRV_S_NETTRACEFILE	このファイルに書き込まれる Net-Library のトレース。	このプロパティの最大長は <b>SRV_MAXCHAR</b> バイト。

プロパティ名	定義	注意
SRV_S_NUMCONNECTIONS	Open Server アプリケーションが受け入れる物理ネットワーク接続の最大数。	サブチャンネルがいくつ使用されるかに関係なく、サーバからサーバへの接続は物理接続1つだけである。たとえば、パススルー Open Server アプリケーション内の送信クライアント・ライブラリ接続は、CS_MAX_CONNECT プロパティにより制限される。CS_MAX_CONNECT は、ct_config() を使用して設定できる。
SRV_S_NUMLISTENERS	クライアント接続の受信に関連する SRV_PROC 制御構造体の数を返す。	これは、取得だけが可能なプロパティである。
SRV_S_NUMMSGQUEUES	Open Server アプリケーションに対して使用できるメッセージ・キューの数。	
SRV_S_NUMMUTEXES	Open Server アプリケーションに対して使用できる相互排他セマフォの数。	
SRV_S_NUMREMBUF	サーバからサーバへの接続で使用されるウィンドウ・サイズ。これは、受信確認が要求される前に論理サブチャンネル上に未処理のまま残っているパケットの最大数を示す。	
SRV_S_NUMREMSITES	一定の時間にアクティブになっているリモート・サーバ・サイト・ハンドラの最大数。	
SRV_S_NUMTHREADS	Open Server アプリケーションに対して使用できるスレッドの最大数。	
SRV_S_NUMUSEREVENTS	アプリケーションが定義できるユーザ・イベントの数。	
SRV_S_PARAM_BATCHING	ブール値。CS_TRUE の場合、Open Server は複数セットのコマンド・パラメータを受け入れることができます。CS_FALSE の場合は、コマンドごとに1セットのコマンド・パラメータのみが受け入れられます。	

プロパティ名	定義	注意
SRV_S_PREEMPT	ブール値。このプロパティが <b>CS_TRUE</b> に設定されている場合、Open Server はプリエンティブ・スケジューリングを使用する。このプロパティが <b>CS_FALSE</b> に設定されている場合、Open Server は非プリエンティブ・スケジューリングを使用する。	すべてのプラットフォームでプリエンティブ・スケジューリングを使用できるわけではない。 <b>srv_capability</b> ルーチンを使用して、プリエンティブ・スケジューリングを使用できるかどうかを調べること。 スレッド・ライブラリとともに <b>SRV_S_PREEMPT</b> を使用した場合、一部の <b>SRV_S_CURTHREAD</b> 機能が無効になる。
SRV_S_REALLOCFUNC	Open Server がメモリを再割り付けするのに使用するルーチンのアドレス。	
SRV_S_REQUEST_CAP	Open Server アプリケーションが受け入れるデフォルトのクライアント要求。	「機能」(25 ページ) を参照してください。
SRV_S_RESPONSE_CAP	Open Server アプリケーションがサポートしているクライアントへのデフォルト応答。	「機能」(25 ページ) を参照してください。
SRV_S_RETPARAMS	実行時にエラーが発生した場合にリターン・パラメータが送信される。	このサーバ・プロパティをデフォルト ( <b>false</b> ) に設定することで、特定のスレッドに対して動作を制限できる。

プロパティ名	定義	注意
SRV_S_SEC_KEYTAB	DCE セキュリティ・ドライバとともに使用するキータブ・ファイル名 (パス名を含む)。	アプリケーションを実行している現在のログイン・ユーザ以外のプリンシパルを指定できる。プロパティ SRV_S_SEC_PRINCIPAL はプリンシパル名を設定する。 DCE ユーティリティ <code>dcecp</code> を使用して、キータブ・ファイルを作成できる。キータブ・ファイルは通常の UNIX ファイルであるため、このファイルにパーミッションを設定してアクセスを制限する必要がある。このファイルは、 Open Server アプリケーションを起動するユーザが読み込み可能なファイルである必要がある。「 <a href="#">セキュリティ・サービス</a> 」(196 ページ) を参照してください。 このプロパティの最大長は SRV_MAXCHAR バイト。
SRV_S_SEC_PRINCIPAL	Open Server アプリケーションのクレデンシャルを得るときに使用するプリンシパル名。 このプロパティのデフォルト値は Open Server アプリケーションのネットワーク名であるが、この名前は <code>srv_init</code> の実行時に指定できる。	このプロパティの最大長は SRV_MAXCHAR バイト。 「 <a href="#">セキュリティ・サービス</a> 」(196 ページ) を参照してください。
SRV_S_SERVERNAME	Open Server アプリケーションの名前。	この名前は、Open Server アプリケーションが稼働しているときにそのアプリケーションを認識するのに使用される。また、 <code>interfaces</code> ファイル内でそのアプリケーションの受信アドレスを検索する場合にも使用される。 このプロパティの最大長は SRV_MAXCHAR バイト。
SRV_S_SSL_CIPHER	CipherSuite 名をカンマで区切ったリスト。	このプロパティの最大長は SRV_MAXCHAR バイト。

プロパティ名	定義	注意
SRV_S_SSL_LOCAL_ID	ファイル内の情報を復号化するために使用する、ファイル名とパスワードが含まれる構造体。	このプロパティの最大長は SRV_MAXCHAR バイト。
SRV_S_SSL_REQUEST_CLIENT_CERT	クライアントが、Open Server アプリケーションにログインするためには証明書を提示することが必要なことを要求する。	
SRV_S_SSL_VERSION	定義されたいずれかの値でなければならない。	定義済みの値 <ul style="list-style-type: none"> <li>CS_SSLVER_20</li> <li>CS_SSLVER_30</li> <li>CS_SSLVER_TLS1</li> </ul> Adaptive Server Enterprise は、デフォルト (CS_SSLVER_TLS1) を使用した接続のみ受け付ける。
SRV_S_STACKSIZE	各スレッドに割り付けられるスタックのサイズ。	
SRV_S_TDSVERSION	Open Server がすべてのクライアント接続のネゴシエーションに使用する TDS のプロトコル・バージョン。	値のリストについては、 <a href="#">「SRV_S_TDSVERSION」(167 ページ)</a> を参照。
SRV_S_TIMESLICE	アクティブなスレッド 1 つが消費するクロック・チックの数。この数に達すると、タイム・スライス・コールバック・ルーチンが呼び出される。	タイム・スライス・コールバックの詳細については、 <a href="#">srv_callback</a> のページを参照。
SRV_S_TRACEFLAG	適切なトレース・タイプ。	フラグのリストについては、 <a href="#">「SRV_S_TRACEFLAG」(167 ページ)</a> を参照。
SRV_S_TRUNCATELOG	ブール値。このプロパティが CS_TRUE に設定されている場合、Open Server は起動中にログ・ファイルをトランケートする。	SRV_S_TRUNCATELOG プロパティは、 <a href="#">srv_init</a> の呼出し後に設定できる。
SRV_S_USERVLANG	ブール値。このプロパティが CS_TRUE に設定されている場合、エラー・メッセージに Open Server アプリケーションのネイティブ言語が使用される。CS_FALSE に設定されている場合は、エラー・メッセージにクライアントの各国言語が使用される。	
SRV_S_VERSION	使用中の Open Server Server-Library の名前、バージョンのリリース年月日、著作権情報が格納されている文字列。	

プロパティ名	定義	注意
SRV_S_VIRTCLKRATE	1 チックあたりのクロック・レート(ミリ秒単位)。	
SRV_S_VIRTIMER	ブール値。このプロパティが CS_TRUE に設定されている場合、仮想タイマが有効になる。このプロパティが CS_FALSE に設定されている場合、仮想タイマは無効になる。	

## SRV\_S\_TDSVERSION

クライアントのログイン処理中に、Open Server はクライアント・アプリケーションとネゴシエートして、TDS バージョンについて合意します。SRV\_S\_TDSVERSION プロパティ値は、Open Server の開始ポイントを決定します。クライアントは、この開始ポイント以下で通信することに合意します。その後で、ログイン処理中に Open Server アプリケーションは SRV\_T\_TDSVERSION スレッド・プロパティを使用して、特定の接続に対する TDS バージョンを再度ネゴシエートできません。詳細については、「スレッド・プロパティ」(168 ページ)を参照してください。

表 2-26 は、このプロパティに有効な値を示したものです。

表 2-26 : SRV\_S\_TDSVERSION の値

SRV_S_TDSVERSION の値	意味
SRV_TDSNONE	認識できない TDS バージョン。
SRV_TDS_4.0	TDS 4.0 からネゴシエーションが開始される。
SRV_TDS_4.0_2	TDS 4.0.2 からネゴシエーションが開始される。
SRV_TDS_4.2	TDS 4.2 からネゴシエーションが開始される。
SRV_TDS_4.6	TDS 4.6 からネゴシエーションが開始される。
SRV_TDS_4.9_5	TDS 4.9.5 からネゴシエーションが開始される。
SRV_TDS_5.0	TDS 5.0 からネゴシエーションが開始される。

## SRV\_S\_TRACEFLAG

SRV\_S\_TRACEFLAG プロパティはビットマップです。このプロパティのフラグは論理和 OR をとることができます。これについては、表 2-27 で説明しています。

表 2-27 : SRV\_S\_TRACEFLAG の値

フラグ	意味
SRV_TR_ATTN	Open Server は、Open Server アプリケーションがアテンションを受信したかまたは受信確認したかを示す情報を表示する。
SRV_TR_DEFQUEUE	Open Server は、イベント・キューのアクティビティをトレースする。
SRV_TR_EVENT	Open Server は、トリガをかけたイベントについての情報を表示する。
SRV_TR_MSGQ	Open Server は、「メッセージ・キュー」のアクティビティをトレースする。
SRV_TR_NETDRIVER	Open Server は、TCL Net-Library ドライバ要求をトレースする。
SRV_TR_NETREQ	Open Server は、TCL 要求をトレースする。
SRV_TR_NETWAKE	Open Server は、TCL ウェイクアップ要求をトレースする。
SRV_TR_TDSDATA	Open Server は、TDS パケットの内容を 16 進数または ASCII 形式で表示する。これは、クライアントと Open Server アプリケーションとの間の実際の TDS トラフィックである。
SRV_TR_TDSHDR	Open Server は、パケットのタイプや長さなどの TDS プロトコル・パケットのヘッダ情報を表示する。

## スレッド・プロパティ

スレッドとは、特定のタスクまたは一連のタスクを遂行するために実行する 1 つのコードです。Open Server スレッドにはいくつかのタイプがあります。スレッド・プロパティはスレッドの動作のさまざまな内容を定義し、そのリソースに制限を設定します。

Open Server スレッドの詳細については、「[マルチスレッド・プログラミング](#)」(123 ページ)を参照してください。

設定できるのはごくわずかのスレッド・プロパティですが、すべてのスレッド・プロパティを取得できます。アプリケーションは、`srv_thread_props` を呼び出して、スレッド・プロパティ値の取得と設定を行います。設定できるプロパティについては、`srv_thread_props` のページを参照してください。アプリケーションは、初期化の後ではいつでもスレッド・プロパティの取得と設定ができます。



Open Server は、初期化時にスレッドを作成するときに、設定できるそれぞれのスレッド・プロパティに対してデフォルトを割り当てます。デフォルトのリストについては、「[srv\\_thread\\_props](#)」(485 ページ)を参照してください。

表 2-28 : スレッド・プロパティ

プロパティ名	定義	注意
SRV_T_APPLNAME	クライアント・アプリケーションの名前。	
SRV_T_BYTEORDER	クライアントが要求したバイト順スキーム。SRV_LITTLE_ENDIAN は、最小有効バイトが上位バイトであることを示す。SRV_BIG_ENDIAN は、最小有効バイトが下位バイトであることを示す。	
SRV_T_BULKTYPE	クライアントが送信するバルク転送のタイプ。	有効な値のリストについては、「 <a href="#">SRV_T_BULKTYPE</a> 」(177 ページ)を参照。
SRV_T_CHARTYPE	文字データ表現のタイプ。	有効な値のリストについては、「 <a href="#">SRV_T_CHARTYPE</a> 」(178 ページ)を参照。
SRV_T_CIPHER_SUITE	CS_CHAR* CipherSuite は、SSL ベースのセッション中に交換されるデータの暗号化や暗号の解読に使用する。 CipherSuite は、接続ハンドシェイク中にネゴシエートされる。	このプロパティは、SRV_LISTENER_PREBIND Open Server イベント内から設定できます。
SRV_T_CLIB	Open Server アプリケーションに接続するのにクライアントが使用するライブラリ製品の名前。	
SRV_T_CLIBVERS	Open Server アプリケーションへの接続にクライアントが使用するライブラリ製品のバージョン。	
SRV_T_CLIENTLOGOUT	ブール値。クライアントが正常ログアウトとアポート・ログアウトのどちらを完了したかを示す。CS_TRUE は正常ログアウトを示す。	このプロパティは、SRV_DISCONNECT イベント・ハンドラ内部からのみ取得できる。
SRV_T_CONVERTSHORT	ブール値。4 バイトの datetime データ型、4 バイトの float データ型、4 バイトの money データ型を、8 バイトのデータ型に自動的に変換するかどうかを示す。	

プロパティ名	定義	注意
SRV_T_DUMPLOAD	ブール値。このクライアント接続に対してダンプ/ロードとバルク挿入の使用を禁止するかどうかを示す。	
SRV_T_ENDPOINT	接続されたクライアントのファイル記述子またはファイル処理。サブチャンネルの場合、サイト・ハンドラの終了ポイント値が返される。 SRV_T_ENDPOINT は Client-Library での CS_ENDPOINT 値に相当する。	このプロパティは、クライアント・スレッド、サイト・ハンドラ、サブチャンネルには有効であるが、サービス・スレッドには有効ではない。 SRV_T_ENDPOINT の使用例については、 <a href="#">「SRV_T_ENDPOINT」(178 ページ)</a> を参照。
SRV_T_EVENT	スレッドが現在存在している Open Server イベント。	有効な値のリストについては、 <a href="#">「SRV_T_EVENT」(179 ページ)</a> を参照。
SRV_T_EVENTDATA	Open Server アプリケーションによって発生する特定のイベントと対応する汎用データ・アドレス。	srv_event ルーチンを使用して設定されるデータ・アドレス。
SRV_T_FULLPASSTHRU	ブール値。このプロパティが CS_TRUE に設定されている場合、SRV_FULLPASSTHRU イベント・ハンドラはそのスレッドに対してアクティブになる。	Open Server アプリケーションの接続ハンドラ内部からのみ設定できる。 SRV_T_EVENT プロパティの値は、フル・パスマルター・イベント・ハンドラ内部から取得される場合は SRV_FULLPASSTHRU である。
SRV_T_FLTTYPE	クライアントが使用する浮動小数点表現のタイプ。	有効な値のリストについては、 <a href="#">「SRV_T_FLTTYPE」(180 ページ)</a> を参照。
SRV_T_GOTATTENTION	ブール値。クライアント・スレッドがアテンションを受信したかどうかを示す。	
SRV_T_HOSTNAME	クライアント接続が発生したホスト・マシンの名前。	
SRV_T_HOSTPROCID	クライアント・プログラムのプロセス ID。	これは、クライアントのログイン・レコード内で受信されるオペレーティング・システム・プロセス ID である。

プロパティ名	定義	注意
SRV_T_IODEAD	ブール値。スレッドの I/O チャンネルが有効かどうかを示す。	CS_TRUE は、スレッドで I/O を正常に実行できないことを示し、CS_FALSE は正常に実行できることを示す。Open Server は、サービス・スレッドに対しては必ず CS_FALSE を返す。
SRV_T_LISTENADDR	指定された SRV_PROC 制御構造体で識別されたリスナのアドレスを返す。SRV_PROC 制御構造体がリスナの場合、このプロパティは、リスナが接続を受け入れるアドレスを返す。	このプロパティには、引数として CS_TRANADDR 構造体へのポインタが必要。これは、取得だけが可能なプロパティである。
SRV_T_LOCALE	Open Server アプリケーションが割り付ける CS_LOCALE 構造体へのポインタ。	このプロパティを使用して、ローカライゼーション情報の取得または設定を行うこと。
SRV_T_LOCALID	リスナで使用する SSL 証明書を指定する。	このプロパティを使用すると、リスナはグローバル・サーバ・レベルの SSL 証明書とは異なる SSL 証明書を使用できる。 これは、設定専用プロパティである。
SRV_T_LOGINTYPE	受信されるログイン・レコードのタイプ。	有効な値のリストについては、「SRV_T_LOGINTYPE」(180 ページ)を参照。
SRV_T_MIGRATED	ブール値。接続が新しい接続であるか、マイグレートされた接続であるかを示す。この読み込み専用プロパティは、クライアントがマイグレート中であるか、サーバにマイグレートした場合は true に設定されます。	詳細については、「SRV_T_MIGRATED」(181 ページ)を参照してください。
SRV_T_MIGRATE_STATE	クライアントのマイグレーション・ステータスを示す。どのスレッドでもアクセスできる読み込み専用プロパティ。	詳細については、「SRV_T_MIGRATE_STATE」(181 ページ)を参照してください。
SRV_T_MACHINE	クライアント・スレッドが実行されているマシンのホスト名。	

プロパティ名	定義	注意
SRV_T_NEGLOGIN	クライアントからの要求があった場合に、ネゴシエートされたログインのタイプ。	<p>このプロパティはビットマスクであり、次に示す5つの値のいずれかを取る。</p> <ul style="list-style-type: none"> <li>• <b>SRV_CHALLENGE</b> は、クライアントがチャレンジ/応答の交換によってネゴシエートしようとしていることを通知する。</li> <li>• <b>SRV_ENCRYPT</b> は、クライアントが対称暗号化パスワードを渡そうとしていることを通知する。</li> <li>• <b>SRV_SECLABEL</b> は、クライアントがセキュリティ・ラベルを送信することを示す。</li> <li>• <b>SRV_APPDEFINED</b> は、アプリケーションで定義されたログイン・ハンドシェイクが使用されていることを示す。</li> <li>• <b>SRV_EXTENDED_ENCRYPT</b> は、クライアントが非対称暗号化パスワードを渡そうとしていることを通知する。</li> </ul>
SRV_T_NOTIFYCHARSET	ブール値。使用中の文字セットが変更されたときにクライアントに通知するかどうかを示す。	
SRV_T_NOTIFYDB	ブール値。Transact-SQL コマンド <b>use db</b> の結果をクライアントに通知するかどうかを示す。	
SRV_T_NOTIFYLANG	ブール値。使用中の各国言語が変更されたときにクライアントに通知するかどうかを示す。	
SRV_T_NOTIFYPND	クライアントに配信する必要がある保留中の通知の数。	このプロパティは取得だけが可能である。
SRV_T_NUMRMPWDS	リモート・パスワードの数。	
SRV_T_PACKETSIZE	クライアントとの通信に使用されるネゴシエートされたパケット・サイズ。	パケット・サイズは、ログイン時に透過的にネゴシエートされる。

プロパティ名	定義	注意
SRV_T_PASSTHRU	ブール値。クライアント・スレッドがパススルー・モードで動作しているかどうかを示す。	このプロパティは、アプリケーションの接続ハンドラ内部から設定できる。 このプロパティが <b>CS_TRUE</b> に設定されている場合、 <b>srv_getloginf</b> ルーチンと <b>ct_setloginf</b> ルーチンは、 <b>Open Server</b> の機能に関係なくクライアント接続の機能をネゴシエートする。フル・パススルー・ゲートウェイではさまざまなコマンドと結果のタイプは認識されないので、これは適切な動作である。
SRV_T_PRIORITY	<b>Open Server</b> がスレッドに対して設定する予定の優先順位レベル。	このプロパティは取得だけが可能である。スレッドの優先順位を設定するには、 <b>srv_setpri</b> を呼び出す。
SRV_T_PWD	クライアントがログイン・レコード内に送信したパスワード文字列。	リモート・サーバ接続の場合、このプロパティはリモート・サーバのパスワードを返す。
SRV_T_REMOTEADDR	<b>SRV_PROC</b> ピアのアドレスを返す。このプロパティは、クライアントの <b>SRV_PROC</b> に対してのみ有効である。	このプロパティには、引数として <b>CS_TRANADDR</b> 構造体へのポインタが必要。これは、取得だけが可能なプロパティである。
SRV_T_RETPARAMS	実行時にエラーが発生した場合にリターン・パラメータが送信される。	<b>SRV_S_RETPARAMS</b> が設定されている場合は、RPC のリターン動作がすべてのスレッドに適用される。
SRV_T_RMTCERTIFICATE	<b>CS_SSLCERT</b> * クライアントの証明書を表すポインタ。	
SRV_T_RMTPWDS	<b>SRV_RMTPWD</b> の配列。	この構造体の定義については、「 <b>SRV_T_RMTPWDS</b> 」(182 ページ) を参照。
SRV_T_RMTSERVER	クライアント接続の場合は、ローカル・サーバ名。サーバからサーバへの接続の場合は、リモート・サーバ名。	

プロパティ名	定義	注意
SRV_T_ROWSENT	このイベントでクライアントに返されるローの数。	
SRV_T_SEC_CHANBIND	このスレッドと対応するクライアント/サーバ接続で、チャンネル・バインドが使用されているかどうかを示すブール値。	
SRV_T_SEC_CONFIDENTIALITY	このスレッドと対応するクライアント/サーバ接続で、データ機密保持サービスが使用されているかどうかを示すブール値。	このプロパティは通常、データの暗号化技術を使用して実装される。
SRV_T_SEC_CREDTIMEOUT	このスレッドと対応するクライアント/サーバ接続で、クレデンシャルの有効期限が切れるまでの残り時間(秒数)。	次のいずれかの値を取る。 <ul style="list-style-type: none"> <li>• <b>CS_NO_LIMIT</b> — 制限時間はなく、有効期限が切れることはない。</li> <li>• <b>CS_UNEXPIRED</b> — まだ有効期限が切れていない。</li> <li>• <b>0</b> — 有効期限が切れた。</li> <li>• 正の数 — 有効期限が切れるまでの残り時間(秒数)。</li> </ul>
SRV_T_SEC_DATAORIGIN	このスレッドと対応するクライアント/サーバ接続で、データ・オリジン・サービスが使用されているかどうかを示すブール値。	
SRV_T_SEC_DELEGATION	クライアントによって委任が有効になっているかどうかを示すブール値。	このスレッドで行われるすべての作業は、クライアントの権限レベルを使用する必要がある。プリンシパル名にアクセスするには、 <b>SRV_T_USER</b> プロパティを使用すること。別のセキュリティ・ピアを使用してセキュリティ・セッションを開始するときに使用する委任クレデンシャルを得るには、 <b>SRV_T_SEC_DELEGCRE</b> プロパティを使用すること。

プロパティ名	定義	注意
SRV_T_SEC_DELEGCREC	現在のセキュリティ・セッションでのクライアントの委任クレデンシヤル(存在する場合)。	SRV_T_SEC_DELEGATIONプロパティは、クライアントによって委任が有効になっているかどうかを示す。委任が有効になっている場合、Open Server アプリケーションはSRV_T_SEC_DELEGCRECプロパティを使用して、委任クレデンシヤルを得ることがある。
SRV_T_SEC_DETECTREPLAY	このスレッドに関連付けられたクライアント/サーバ接続で、メッセージ・リプレイの検出が使用されているかどうかを示すブール値。	
SRV_T_SEC_DETECTSEQ	このスレッドと対応するクライアント/サーバ接続で順序不整合の検出のメッセージ・サービスが使用されているかどうかを示すブール値。	
SRV_T_SEC_INTEGRITY	このスレッドと対応するクライアント/サーバ接続で、整合性サービスが使用されているかどうかを示すブール値。	通常このプロパティは、暗号化シングニチャを使用して実装される。
SRV_T_SEC_MECHANISM	このスレッドと対応するクライアント/サーバ接続で使用されているセキュリティ・メカニズムのローカル名。	
SRV_T_SEC_MUTUALAUTH	このスレッドと対応するクライアント/サーバ接続で、相互認証が実行されたかどうかを示すブール値。	
SRV_T_SEC_NETWORKAUTH	このスレッドと対応するクライアント/サーバ接続で、ネットワーク認証が実行されたかどうかを示すブール値。	

プロパティ名	定義	注意
SRV_T_SEC_SESTIMEOUT	このスレッドと対応するクライアント/サーバ接続で、セキュリティ・セッションの有効期限が切れるまでの残り時間 (秒数)。	次のいずれかの値を取る。 <ul style="list-style-type: none"> <li>CS_NO_LIMIT - 制限時間はなく、有効期限が切れることはない。</li> <li>CS_UNEXPIRED - まだ有効期限が切れていない。</li> <li>0 - 有効期限が切れた。</li> <li>正の数 - 有効期限が切れるまでの残り時間 (秒数)。</li> </ul>
SRV_T_SESSIONID	クライアントが Open Server に送信するセッション ID を取得する。さらに、セッション ID がクライアントに送信されるように SRV_CONNECT ハンドラで設定する。	詳細については、「SRV_T_SESSIONID」(183 ページ) を参照してください。
SRV_T_SSL_VERSION	接続ハンドシェイク中にネゴシエートされた SSL/TLS プロトコル・バージョン。	このプロパティは、SRV_LISTENER_PREBIND Open Server イベント内から設定できます。
SRV_T_SPID	スレッドのプロセス ID。	このプロパティは、このスレッドに割り当てられたユニークな ID である。スレッドが終了すると、スレッド ID は再使用される。
SRV_T_STACKLEFT	スレッドに使用できる未使用スタックのサイズ。	
SRV_T_TDSVERSION	クライアント・スレッドが使用している TDS のバージョン。	SRV_CONNECT イベント・ハンドラ内でこのスレッドを設定すると、Open Server アプリケーションは、そのスレッドに対する Open Server のデフォルト以外の値に TDS バージョンをネゴシエートできる。有効な値のリストについては、「SRV_T_TDSVERSION」(183 ページ) を参照。
SRV_T_TYPE	スレッド・タイプ。	有効な値のリストについては、「SRV_T_TYPE」(184 ページ) を参照。



プロパティ名	定義	注意
SRV_T_USER	クライアント・スレッドがログインしたときに使用したユーザ名。	
SRV_T_USERDATA	アプリケーションごとに異なる目的で使用される汎用データ・アドレス。	このプロパティは設定可能である。
SRV_T_USESRVLANG	ブール値。サーバの各国言語でエラー・メッセージを表示させる場合には、このプロパティを <b>CS_TRUE</b> に設定する。クライアントの各国言語でエラー・メッセージを表示する場合には、このプロパティを <b>CS_FALSE</b> に設定する。	スレッドのサーバワイドな <b>SRV_S_USESRVLANG</b> プロパティを上書きするには、このプロパティを設定する。
SRV_T_USTATE	スレッドの現在のステータスを記述する文字列。	このプロパティは設定可能である。

## SRV\_T\_BULKTYPE

クライアント・アプリケーションは、つぎの3つのタイプのバルク・データ、バルク・コピー・データ、テキスト・データ、およびイメージ・データを Open Server アプリケーションに転送できます。クライアントによって開始されるバルク・データ転送のタイプの設定または取得を行うには、SRV\_T\_BULKTYPE プロパティを使用します。

表 2-29 は、SRV\_T\_BULKTYPE スレッド・プロパティの有効値をまとめたものです。

**表 2-29 : SRV\_T\_BULKTYPE の値**

値	意味
SRV_BULKLOAD	クライアントはバルク・コピー・データを転送する準備をしている。
SRV_TEXTLOAD	クライアントは text データを転送する準備をしている。
SRV_IMAGELOAD	クライアントは image データを転送する準備をしている。
SRV_UNITEXTLOAD	クライアントは unitext データを転送する準備をしている。

Open Server は、クライアントが送信するバルク・データ・ストリームのタイプを自動的に決定できません。Open Server アプリケーションは `srv_thread_props` ルーチンを使用してこの情報を取得し、実際の SRV\_BULK イベントより前にその情報を Open Server に提供する必要があります。その後アプリケーションは、一度実際のバルク要求が行われていれば、SRV\_BULK イベント・ハンドラ内部からデータを取得します。

『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。「[text と image](#)」(225 ページ)を参照してください。

## SRV\_T\_CHARTYPE

クライアント・アプリケーションは、文字データが特定の方法で表現されると仮定します。クライアントが想定している文字データ表現方法を Open Server アプリケーション側で取得するには、`property` を `SRV_T_CHARTYPE` に設定し、`cmd` を `CS_GET` に設定して `srv_thread_props` を呼び出します。クライアントは `*bufp` で次の値を返します。

**表 2-30 : 文字データ表現**

値	意味
SRV_CHAR_ASCII	ASCII 文字フォーマット
SRV_CHAR_EBCDIC	EBCDIC 文字フォーマット
SRV_CHAR_UNKNOWN	認識できない文字フォーマット

## SRV\_T\_ENDPOINT

次の例は、SRV\_T\_ENDPOINT の使用方法を示します。

```
CS_INT ep;
/*
```

```
** Get the end point
*/
if(srv_thread_props(spp, CS_GET, SRV_T_ENDPOINT, (CS_VOID *)&ep,
    CS_SIZEOF(ep), (CS_INT *)NULL) == CS_FAIL)
{
    return(CS_FAIL);
}
```

## SRV\_T\_EVENT

スレッドは一度に1つの特定のイベント・ハンドラを実行します。イベントに対応するイベント・ハンドラを実行するときに、スレッドはそのイベントの内部にあると言えます。Open Server アプリケーションは、property を SRV\_T\_EVENT に設定し、cmd を CS\_GET に設定して `srv_thread_props` を呼び出し、スレッドが内部にあるイベントを取得できます。アプリケーションが複数のイベントに対して同じイベント・ハンドラ・コードを使用する場合は、この手順が便利です。

このようなイベントには、次のものがあります。

- SRV\_ATTENTION
- SRV\_BULK
- SRV\_CONNECT
- SRV\_CURSOR
- SRV\_DISCONNECT
- SRV\_DYNAMIC
- SRV\_FULLPASSTHRU
- SRV\_LANGUAGE
- SRV\_LISTENER\_PREBIND
- SRV\_LISTENER\_POSTBIND
- SRV\_MSG
- SRV\_OPTION
- SRV\_RPC
- SRV\_START
- SRV\_STOP
- ユーザ定義イベント

[「イベント」\(100 ページ\)](#)を参照してください。

## SRV\_T\_FLTTYPER

クライアント・アプリケーションは、浮動小数点データが特別な方法で表現されると仮定します。クライアントが想定している浮動小数点データ表現方法を Open Server アプリケーション側で取得するには、property を SRV\_T\_FLTTYPER に設定し、cmd を CS\_GET に設定して srv\_thread\_props を呼び出します。クライアントは、bufp が示すアドレス領域内に次の値のいずれかを返します。

- SRV\_FLT\_IEEE – IEEE の浮動小数点フォーマット
- SRV\_FLT\_ND5000 – ND5000 の浮動小数点フォーマット
- SRV\_FLT\_VAX – VAX ‘D’ の浮動小数点フォーマット
- SRV\_FLT\_UNKNOWN – 認識できない浮動小数点フォーマット

## SRV\_T\_LOGINTYPE

Open Server アプリケーションは、ログイン処理中にさまざまなタイプのスレッド・ログイン・レコードのすべてを受信できます。

SRV\_T\_LOGINTYPE プロパティはログイン・タイプを示します。アプリケーションは、property を SRV\_T\_LOGINTYPE に設定し、cmd を CS\_GET に設定して srv\_thread\_props を呼び出し、ログイン・タイプを取得できます。このログイン・タイプは、bufp が示すバッファ内に返されます。表 2-31 に、それぞれのログイン・タイプを示します。

表 2-31 : スレッド・ログイン・タイプ

値	ログイン・タイプ
SRV_SITEHANDLER	リモート・サーバからのサイト・ハンドラ・ログイン要求
SRV_SUBCHANNEL	リモート・サーバからのサイト・ハンドラ・サブチャネル・ログイン
SRV_CLIENT	クライアント・アプリケーションからのログイン要求

## SRV\_T\_MIGRATED

接続が新しい接続であるか、マイグレートされた接続であるかを示す Boolean プロパティです。この読み込み専用プロパティは、クライアントがマイグレート中であるか、サーバにマイグレートした場合は true に設定されます。次のサンプル・コードは、SRV\_T\_MIGRATED の値を取得します。

```
CS_RETCODE      ret;
CS_BOOL migrated;
status = srv_thread_props(sp, CS_GET, SRV_T_MIGRATED,
&migrated, sizeof (migrated), NULL);
```

詳細については、「[接続マイグレーション](#)」(41 ページ)を参照してください。

## SRV\_T\_MIGRATE\_STATE

SRV\_T\_MIGRATE\_STATE は、クライアントのマイグレーション・ステータスを示します。どのスレッドでもアクセスできる読み込み専用プロパティ。示されるマイグレーション・ステータスは次のとおりです。

状態	値	説明
SRV_MIG_NONE	0	進行中のマイグレーションがない。
SRV_MIG_REQUESTED	1	マイグレーションがサーバにより要求された。
SRV_MIG_READY	2	クライアントが要求を受信し、マイグレーションの準備ができた。
SRV_MIG_MIGRATING	3	クライアントが指定されたサーバにマイグレート中である。
SRV_MIG_CANCELLED	4	マイグレーション要求がキャンセルされた。
SRV_MIG_FAILED	5	クライアントがマイグレーションに失敗した。

SRV\_MIG\_STATE は、SRV\_T\_MIGRATE\_STATE プロパティをモデル化する列挙データ型です。SRV\_MIG\_STATE は、次のように宣言されます。

```
typedef enum
{
    SRV_MIG_NONE,
    SRV_MIG_REQUESTED,
    SRV_MIG_READY,
    SRV_MIG_MIGRATING,
    SRV_MIG_CANCELLED,
    SRV_MIG_FAILED
} SRV_MIG_STATE;
```

次のサンプル・コードは、`SRV_T_MIGRATE_STATE` 値を取得する方法を示しています。マイグレーションに成功した場合、クライアントが終了し、`SRV_DISCONNECT` イベント・ハンドラが `SRV_MIG_MIGRATING` ステータスで呼び出されます。

```
CS_RETCODE    ret;
SRV_MIG_STATE migration_state;
ret = srv_thread_props(sp, CS_GET, SRV_T_MIGRATE_STATE,
    &migration_state, sizeof (migration_state), NULL);
if (ret != CS_SUCCEED)
{
    ...
}
```

詳細については、「[接続マイグレーション](#)」(41 ページ) を参照してください。

## SRV\_T\_RMTPWDS

アプリケーションは、`SRV_T_RMTPWDS` プロパティを使用してリモート・サーバの名前とパスワードを取得します。この名前とパスワードが格納される `SRV_T_RMTPWD` 構造体の定義は次のとおりです。

```
typedef struct srv_rmtpwd
{
    CS_INT servnamelen;
    CS_BYTEservname[CS_MAX_NAME];
    CS_INTpwdlen;
    CS_BYTEpwd[CS_MAX_NAME];
} SRV_RMTPWD;
```

## SRV\_T\_SESSIONID

SRV\_T\_SESSIONID は、クライアントから Open Server に送信されたセッション ID を取得するスレッド・プロパティです。Open Server アプリケーションは、次の場合に `srv_thread_props()` 関数を使用して SRV\_T\_SESSIONID プロパティを設定することもできます。

- `srv_thread_props(CS_SET, SRV_T_SESSIONID)` 呼び出しが SRV\_CONNECT イベント・ハンドラ内で行われた。
- クライアントで接続マイグレーションまたは高可用性がサポートされている。

次のサンプル・コードは、SRV\_T\_SESSIONID プロパティを設定します。

```
CS_RETCODE    ret;
CS_SESSIONID  hasessionid;
ret = srv_thread_props(sp, CS_SET, SRV_T_SESSIONID,
                      hasessionid, sizeof(hasessionid), NULL);
```

## SRV\_T\_TDSVERSION

クライアントのログイン処理中に、Open Server はクライアント・アプリケーションとネゴシエートして、すべてのスレッドに対してある TDS バージョンで合意します。SRV\_S\_TDSVERSION プロパティ値は、Open Server の開始ポイントを決定します。クライアントは、この開始ポイント以下で通信することに合意します。

SRV\_S\_TDSVERSION プロパティの詳細については、「[スレッド・プロパティ](#)」(168 ページ)を参照してください。その後で、ログイン処理中に Open Server アプリケーションは SRV\_T\_TDSVERSION プロパティを使用して、特定のスレッドに対して TDS バージョンを再度ネゴシエートできます。

表 2-32 は、このプロパティに有効な値を示したものです。

表 2-32 : SRV\_T\_TDSVERSION の値

SRV_T_TDSVERSION の値	意味
SRV_TDSNONE	認識できない TDS バージョン。
SRV_TDS_4.0	TDS 4.0 からネゴシエーションが開始される。
SRV_TDS_4_0_2	TDS 4.0.2 からネゴシエーションが開始される。
SRV_TDS_4_2	TDS 4.2 からネゴシエーションが開始される。
SRV_TDS_4_6	TDS 4.6 からネゴシエーションが開始される。
SRV_TDS_4_9_5	TDS 4.9.5 からネゴシエーションが開始される。
SRV_TDS_5_0	TDS 5.0 からネゴシエーションが開始される。

## SRV\_T\_TYPE

Open Server スレッドにはいくつかのタイプがあります。SRV\_T\_TYPE スレッド・プロパティは、スレッドのタイプを示します。アプリケーションでスレッドのタイプを取得するには、property を SRV\_T\_TYPE に設定し、cmd を CS\_GET に設定して `srv_thread_props` を呼び出します。

表 2-33 は、有効なスレッド・タイプを示したものです。

表 2-33 : スレッドのタイプ

値	スレッド・タイプ
SRV_TCLIENT	クライアント・スレッド
SRV_TLISTENER	クライアント接続を受け入れるスレッド
SRV_TSITE	サイト・ハンドラ・スレッド
SRV_TSUBPROC	サイト・ハンドラ・スレッド上のリモート・サーバ接続
SRV_TSERVICE	サービス・スレッド

「マルチスレッド・プログラミング」(123 ページ) を参照してください。



## レジスタード・プロシージャ

レジスタード・プロシージャとは、名前によって識別されたコードです。アプリケーションはプロシージャを登録するときに、プロシージャ名をルーチンにマップします。これにより、Open Server は、受信 RPC データ・ストリーム内でそのプロシージャ名を検出すると、SRV\_RPC イベントを発生させずに特定のルーチンをただちに呼び出すことができます。

Open Server が RPC を受け取ると、Open Server はレジスタード・プロシージャのリストでプロシージャ名を探します。名前が登録されている場合、ランタイム・システムは、存在するレジスタード・プロシージャと関連しているルーチンすべてを実行します。プロシージャ名がレジスタード・プロシージャのリストに見つからない場合には、Open Server は SRV\_RPC イベント・ハンドラを呼び出します。

## 標準リモート・プロシージャ・コール

Open Server アプリケーションは、通常の RPC を、アプリケーションの SRV\_RPC イベント・ハンドラ内から処理します。ハンドラ・コードは、RPC データ・ストリームを解析して、このプロセスの RPC 名、パラメータの数、パラメータのフォーマット、パラメータ値を取得しなければなりません。その後、ハンドラはこれらの値に基づいてアクションを起こすことができます。SRV\_RPC イベント・ハンドラは、ネットワーク経由で送られてくると考えられるすべての RPC を処理できるようにコーディングされている必要があります。

## レジスタード・プロシージャの利点

次の理由により、レジスタード・プロシージャを利用すると、Open Server アプリケーションでの RPC 処理が単純化されます。

- レジスタード・プロシージャは、コードを 1 か所に統合します。レジスタード・プロシージャは、Open Server アプリケーションが SRV\_RPC イベント・ハンドラに加えて、他のイベント・ハンドラから呼び出すことができる実行可能オブジェクトです。

- レジスタード・プロシージャは、**Server-Library** 呼び出しまたは外部の **Client-Library** 呼び出しや **DB-Library** 呼び出しによって、サーバが稼動しているときであればいつでも作成できます。それに対して、**SRV\_RPC** イベント・ハンドラは、サーバの起動の前からかじめコーディングされていなければなりません。
- レジスタード・プロシージャには自動データ型チェック機能があるので、**Open Server** アプリケーション・コードの解析は必要ありません。
- クライアントは、レジスタード・プロシージャの実行時に通知 (ノティフィケーション) を要求できます。この「通知」は、次の要素で構成されます。
  - レジスタード・プロシージャ名
  - レジスタード・プロシージャの実行に関連するパラメータ値
- 通知要求は、**Server-Library** 呼び出しにより内部的に、または **Client-Library** または **DB-Library** 呼び出しにより外部的に発行できます。
- クライアントは、レジスタード・プロシージャのリストや自分が通知を要請したプロシージャのリストを要求することができます。

## ノティフィケーション・プロシージャ

プログラマが提供するコードがない状態で、**Open Server** アプリケーションは、**Client-Library** または **DB-Library** クライアントがレジスタード・プロシージャを作成し、それを実行し、その実行の通知 (ノティフィケーション) を受け取ることができるようにします。

レジスタード・プロシージャは、**Open Server** アプリケーションに実行可能ルーチンを持っている必要はありません。実際、**DB-Library** や **Client-Library** 呼び出しで作成されたレジスタード・プロシージャは、**Open Server** のルーチンを読み出すことができません。実行可能なルーチンと対応付けられていないレジスタード・プロシージャは、実行を待機しているクライアントへの通知が唯一の目的なので、「ノティフィケーション (通知) プロシージャ」と呼ばれます。

ノティフィケーション・プロシージャを使用すれば、クライアント・アプリケーションはどの **Open Server** アプリケーションによっても相互に通信できます。

この機能を使用可能にするためにコードを書く必要はありませんが、レジスタード・プロシージャを使用不可にしたり制限したりするために、コールバック・ハンドラをインストールすることができます。詳細については、「[レジスタード・プロシージャでのコールバック・ハンドラの使用](#)」(190 ページ)を参照してください。

## レジスタード・プロシージャの作成

Open Server アプリケーションは、標準レジスタード・プロシージャとノーティフィケーション・プロシージャの両方を作成することができます。Client-Library と DB-Library アプリケーションは、ノーティフィケーション・プロシージャを作成することができます。Client-Library ルーチンを使ってレジスタード・プロシージャを作成する方法については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

## レジスタード・プロシージャのメカニズム

この項では、Open Server アプリケーション内でレジスタード・プロシージャを作成して実行する方法について説明します。

### レジスタード・プロシージャ

Open Server 呼び出しによってプロシージャを登録するには、次の手順に従います。

- 1 プロシージャ名を定義し、プロシージャが実行されるときに呼び出される関数に名前をマップするために、`srv_regdefine` を呼び出します。
- 2 定義されるプロシージャのパラメータを記述するために、`srv_regparam` を呼び出します。
- 3 プロシージャの登録を完了するために、`srv_regcreate` を呼び出します。
- 4 プロシージャの登録を解除するために、`srv_regdrop` を呼び出します。

## レジスタード・プロシージャの実行

RPC が登録されている場合には、Open Server はクライアントまたはリモート Adaptive Server Enterprise による RPC に応答してレジスタード・プロシージャを実行します。しかし、Open Server アプリケーションは、RPC への応答の中でそれを実行する代わりに、レジスタード・プロシージャを明示的に実行することができます。たとえば、アプリケーションが、特定のノーティフィケーション・プロシージャをアプリケーションの特定の時点で実行することによって、複数のクライアントのアクティビティを同期できます。

レジスタード・プロシージャを明示的に実行するにも、いくつかの手順が必要です。それらは、次のとおりです。

- 1 レジスタード・プロシージャの実行を開始するために、`srv_reginit` を呼び出します。このルーチンは、実行されるレジスタード・プロシージャの名前を指定します。Open Server アプリケーションは、通知リストのクライアント・スレッドの 1 つが通知されるのか、またはすべてが通知されるのかを調べるためにも、このルーチンを使用します。
- 2 実行のためのパラメータ・データを提供するために、`srv_regparam` を呼び出します。
- 3 レジスタード・プロシージャを実際に実行するために、`srv_regexec` を呼び出します。

## リストの管理

Open Server アプリケーションは、すべてのレジスタード・プロシージャのリストと、特定のレジスタード・プロシージャが実行されたときに通知するクライアントのリストを管理しています。この通知は、自動的に起こります。リストの管理に関連しているルーチンは次のとおりです。

- `srv_reglist` — Open Server アプリケーション内で登録されているすべてのプロシージャのリストを返します。
- `srv_regwatchlist` — 指名されたクライアント・スレッドが通知要求待ちを示す、対象となるレジスタード・プロシージャのすべてのリストを返します。
- `srv_regwatch` — レジスタード・プロシージャの通知リストにスレッドを追加します。
- `srv_regnowatch` — 特定のレジスタード・プロシージャの通知リストからクライアントを削除します。

- `srv_reglistfree` - `srv_reglist` または `srv_regwatchlist` によって割り付けられた `SRV_PROCLIST` 構造体を解放します。

## システム・レジスタード・プロシージャ

各 Open Server アプリケーションは、「システム・レジスタード・プロシージャ」と呼ばれる組み込みレジスタード・プロシージャを持っています。これらは、サーバが起動する時点でランタイム・システムが作成します。システム・レジスタード・プロシージャについては、「[第4章 システム・レジスタード・プロシージャ](#)」で説明されています。これらのプロシージャの中には、Open Server アプリケーションを対話型で管理するために役立つものもあります。たとえば、`sp_who` と `sp_ps` を使用すると、アクティブなサーバ・プロセスをリストできます。また、`sp_terminate` を使用すると、プロセスを消滅させることができます。

クライアント・アプリケーションでは、システム・レジスタード・プロシージャを実行することで、次の処理を行うことができます。

- レジスタード・プロシージャのリストの取得
- レジスタード・プロシージャを実行する
- レジスタード・プロシージャの実行の通知の要求
- 通知要求のリストの取得

ほとんどのシステム・レジスタード・プロシージャは、同等の Open Server ルーチンに対応します。Open Server アプリケーションとクライアントは、異なるルーチンによって同じ種類の情報を要求できます。

[表 2-34](#) は、システム・レジスタード・プロシージャと対応する Server-Library ルーチンをまとめたものです。

表 2-34 : システム・レジスタード・プロシージャと対応する Server-Library ルーチン

システム・レジスタード・プロシージャ	Server-Library ルーチン
sp_ps	N/A
sp_regcreate	srv_regcreate/srv_regdefine
sp_regdrop	srv_regdrop
sp_reglst	srv_reglst
sp_regnowatch	srv_regnowatch
sp_regwatch	srv_regwatch
sp_regwatchlist	srv_regwatchlist
sp_serverinfo	N/A
sp_terminate	srv_termproc
sp_who	N/A

## レジスタード・プロシージャでのコールバック・ハンドラの使用

表 2-34 に示されているように、組み込みレジスタード・プロシージャのいくつかは、レジスタード・プロシージャを作成、削除、実行する Server-Library ルーチンおよび DB-Library ルーチンに相当します。これらのプロシージャは、レジスタード・プロシージャが実行しようとするときにいつでも実行するようなコールバック・ハンドラをインストールすることによって、レジスタード・プロシージャのためのセキュリティ・システムを実装できるようにします。クライアント・アプリケーションがシステム・レジスタード・プロシージャまたはそれに相当する Client-Library や DB-Library ルーチンのひとつを実行すると、コールバック・ハンドラが実行されます。SRV\_S\_INHIBIT を返した場合は、レジスタード・プロシージャは実行されません。

たとえば、"sa" 以外のクライアントが "reinitialize" という名前のプロシージャを実行できないようにするには、レジスタード・プロシージャ・コールバック・ハンドラに次のコードを含めます。

```

/*
** Stop users other than "sa" from executing the "reinitialize"
** registered procedure.
**
** Parameters:
** spp - Handle to the current client connection.
**
** Returns:
** CS_TRUE Allow the user to execute

```

```
** CS_FALSE Disallow execution.
*/
CS_BOOL   rpc_permission(spp)
SRVPROC   *spp;
{
    CS_INT ulen;          /* User name length */
    CS_INT rlen;         /* RPC name length */
    CS_CHAR *rname;      /* Pointer to the RPC name */
    CS_CHAR user[256];   /* Buffer for the user name */

    /*
    ** Get the name of the rpc command
    */
    if ((rname = srv_rpcname(spp, &rlen)) == (CS_CHAR *)NULL)
    {
        return (CS_FALSE);
    }

    /*
    ** Get the user name.
    */
    if (srv_thread_props(spp, CS_GET, SRV_T_USER,
        (CS_VOID *)user, CS_SIZEOF(user), &ulen) == CS_FAIL)
    {
        return (CS_FALSE);
    }

    /*
    ** If either the user name or the rpc name is NULL,
    ** indicate an error.
    */

    if (rlen <= 0 || ulen <= 0)
    {
        error ("API error");
        return (CS_FALSE);
    }

    /* Null terminate the user name buffer */
    user[ulen] == '¥0';

    /*
    ** Compare the RPC name and User name for permission.
    */
    if ((strcmp(rname, "reinitialize") == 0) &&
        (strcmp(user, "sa") == 0))
    {
```

```
return (CS_TRUE);  
}  
  
return (CS_FALSE);  
}
```

## 例

サンプル・プログラム *regproc.c* には、Open Server アプリケーションによるレジスタード・プロシージャの使用例が記述されています。

## リモート・プロシージャ・コール

リモート・プロシージャ・コール (RPC) は、クライアント・アプリケーションが Open Server アプリケーションと通信するためのメカニズムです。通常は、クライアントは、Open Server アプリケーションから情報を得るために RPC を発行します。RPC は、名前と、多くの場合 (必ずとは限らない) パラメータから構成されています。たとえば、デパートのアプリケーションは、`get_cust` という RPC への応答として顧客の氏名と住所を返します。この RPC は、1 つのパラメータ、つまり顧客 ID 番号を受け取ります。

クライアントが RPC を送ると、Open Server はその RPC が登録済みかどうかをチェックします。レジスタード・プロシージャは、Open Server がアプリケーションの `SRV_RPC` イベント・ハンドラを呼び出すことなく直接認識して実行する、特殊な RPC です。「[レジスタード・プロシージャ](#)」(185 ページ) を参照してください。

RPC が登録されていない場合、Open Server は `SRV_RPC` イベントをトリガします。`SRV_RPC` イベント・ハンドラ内から、アプリケーションは RPC の名前とパラメータ (ある場合) を取得し、適切に応答することができます。イベント・ハンドラは、クライアントから送られる可能性のあるすべての RPC の名前と、それぞれが持つパラメータの数を検証するようにコーディングされています。ハンドラには、個々の RPC に応答するためのコードが含まれており、RPC を認識できない場合は、クライアントにエラー情報を返します。

`SRV_RPC` イベント・ハンドラの中では、アプリケーションは次の手順を実行します。



- 1 RPC名を取得するために、`srv_rpcname` を呼び出します (アプリケーションは、`srv_rpcnumber`、`srv_rpcowner`、`srv_rpcdb` を使用して、RPC番号、所有者、関連データベースをそれぞれ取得することもできます)。該当するRPCが存在しない場合や、番号、所有者、またはデータベースの情報が無効である場合には、アプリケーションは `srv_sendinfo` によってエラー情報を返します。
- 2 `srv_numparams` を呼び出して、適切な数のパラメータが送信されたことを検証します。パラメータ情報に無効なものがあれば、`srv_sendinfo` によってエラー情報を返します。
- 3 `srv_descfmt`、`srv_bind`、`srv_xferdata` を呼び出して、パラメータを処理します。詳細については、「[パラメータとロー・データの処理](#)」(152 ページ) を参照。
- 4 `srv_descfmt`、`srv_bind`、`srv_xferdata` を呼び出して、データをクライアントに返します。詳細については、「[パラメータとロー・データの処理](#)」(152 ページ) を参照してください。

RPC パラメータは、名前または位置で受け渡されます。一部のパラメータが名前で、他のパラメータが位置で受け渡されるような状況でRPCが呼び出された場合には、エラーが発生します。

アプリケーションは、すべてのプロシージャを登録し、`SRV_RPC` イベント・ハンドラを使用してエラーをトラップすることもできます。この場合には、クライアントが無登録の、つまり無効なRPCを送った場合にのみ `Open Server` が `SRV_RPC` イベント・ハンドラを呼び出すこととなります。これを受けて `SRV_RPC` イベント・ハンドラが `srv_sendinfo` を使って、クライアントに対して無効なRPCを発行したことを通知します。

## 例

サンプル・プログラム `regproc.c` には、リモート・プロシージャ・コールの例が記述されています。

## ラージ・オブジェクト・パラメータの取得

Open Server は、`srv_xferdata()` を使用してラージ・オブジェクト (LOB) パラメータ・データ全体を同時に取得したり、`srv_get_data` を使用してまとまりで取得することができます。Open Server は、パラメータの長さが `CS_UNUSED` に設定されている場合に、LOB パラメータをまとまりで取得します。「[srv\\_get\\_data](#)」(320 ページ) を参照してください。

例 LOB パラメータの説明を取得します。

```
/*
** Retrieve the description of the parameters coming
** from client
*/

for (paramnum = 1; paramnum <= numparams; paramnum++)
{
    /*
    ** Get a description of the parameter.
    */
    ret = srv_descfmt(spp, CS_GET, SRV_RPCDATA,
        paramnum, &(paramfmt[paramnum - 1]));

    /*
    ** Allocate space for the parameters and bind the
    ** data.
    */
    if (paramfmt[paramnum-1].maxlength >= 0)
    {
        if (paramfmt[paramnum-1].maxlength > 0)
        {
            data[paramnum-1] = calloc(1,
                paramfmt[paramnum-1].maxlength);
        }
        else
        {
            ind[paramnum-1] = CS_NULLDATA;
        }
    }
    else
    {
        /*
        ** Allocate a large size buffer for BLOB data
        ** (which length is unknown yet)
        */
        blobbuf[blobnum] = malloc(BUFSIZE);
        blobnum++;
    }
}
```

```
    }

    srv_bind(spp, CS_GET, SRV_RPCDATA, paramnum,
             &(paramfmt[paramnum-1]), data[paramnum-1],
             &(len[paramnum-1]), &(ind[paramnum-1]))

    /*
    ** For every LOB parameter, call srv_get_data() in
    ** a loop as long as it succeeds
    */
    for (i = 0; i < blobnum ; i++)
    {
        bufp = blobbuf[i];
        bloblen[i] = 0;
        do
        {
            ret = srv_get_data(spp, bufp, BUFSIZE,

                               &outlen);
            bufp += outlen;
            bloblen[i] += outlen;
        } while (ret == CS_SUCCEED);

        /*
        ** Check for the correct return code
        */
        if (ret != CS_END_DATA)
        {
            return CS_FAIL;
        }
    }

    /*
    ** And receive remaining client data srv_xferdata()
    */
    ret = srv_xferdata(spp, CS_GET, SRV_RPCDATA);
}
```

## セキュリティ・サービス

セキュリティ・サービスを利用すると、Open Server アプリケーションで、サード・パーティの分散セキュリティを使用してユーザを認証し、クライアントとサーバとの間で転送されるデータを保護することができます。

使用しているプラットフォームで使用可能な分散セキュリティ・サービス・プロバイダについては、プラットフォーム用の『Open Client/Server 設定ガイド』を参照してください。

特定のプロバイダから入手できるセキュリティ・サービスを、**セキュリティ・メカニズム**と呼びます。Open Server アプリケーションは、入手可能かどうかによって複数の「セキュリティ・メカニズム」をサポートできます。Open Server アプリケーションは、クライアントまたはサーバ・ダイアログごとに(クライアント接続要求に基づいて)セキュリティ・メカニズムを選択します。

Open Server のセキュリティ・サービスを使用して、次のタスクを実行できます。

- システム上に確立されている**クレデンシャル**にアクセスできません。

クレデンシャルとは、ピアの ID を確立するためにピア(クライアントとサーバ)間で転送されるデータです。

- ダイアログの確立中に要求されたセキュリティ・メカニズムと通信できます。
- リモート・クライアントまたはリモート・サーバとのセキュリティ・セッションを確立できます。

セキュリティ・サービスは、セキュリティ・セッションの確立中にネゴシエートされます。セキュリティ・セッションは、直接クライアント・ダイアログにマップされます。

- 内部が隠されたトークンをダイアログ上で通信することで、ピア・コンポーネントとの通信でセキュリティ・メカニズムを使用できます。これらのトークンはセッションの確立中に送信され、必要に応じて、パケットごとのセキュリティ・サービスに使用できます。

トークンとは、ピア間でのセキュリティ情報交換のためにセキュリティ・メカニズムが生成するビット文字列です。トークンは、暗号化されて保護される場合もあります。

- セキュリティ・セッションにチャネル識別情報をバインドできません。
- トークンの発生元を保証するために、トークンをデジタル化処理できます。

## セキュリティ・サービス・プロパティ

ネットワーク・セキュリティ・サービスは、大きく分けて次の3つのタイプに分類できます。

- ログイン認証サービス
- パケットごとのセキュリティ・サービス
- SSL (Secure Sockets Layer) 暗号化

## ログイン認証サービス

基本的なセキュリティ・サービスは「ログイン認証」で、ユーザが本人であることを確認するものです。ログイン認証には、ユーザ名とパスワードが必要です。ユーザはユーザ名によってユーザ自身を識別し、そのユーザ本人であることの証拠としてパスワードを入力します。

**Sybase** アプリケーションでは、クライアントとサーバの間の接続ごとに対応するユーザ名が1つずつあります。アプリケーションがセキュリティ・メカニズムを使用する場合、**Sybase** はそのメカニズムを使用して、接続が確立されるときにこのユーザ名を認証します。このサービスの利点は、ユーザ名とそのパスワードを個々のサーバのシステム・カタログ内ではなく中央レポジトリ内で管理できることです。

アプリケーションがネットワークベースの認証を使用してサーバへの接続を要求する場合、**Client-Library** はその接続のセキュリティ・メカニズムに問い合わせ、指定されたユーザ名が認証ユーザを示していることを確認します。つまり、ユーザはパスワードを指定しなくてもサーバに接続できます。その代わりに、ユーザはネットワーク・セキュリティ・システムに対してユーザ自身を認証した後で、接続しようとしています。接続時に、**Client-Library** はセキュリティ・メカニズムから「クレデンシャル・トークン」を取得して、このトークンをパスワードの代わりにサーバに送信します。その後サーバはそのトークンを再度セキュリティ・メカニズムに渡して、そのユーザ名が認証されていることを確認します。

次の表は、ログイン認証に関係のあるプロパティを示します。

**表 2-35 : ログイン認証を制御するプロパティ**

プロパティ	説明
CS_USERNAME	接続時に使用するユーザ名を指定する。
CS_SEC_NETWORKAUTH	ネットワークベースのユーザ認証を有効にする。
CS_SEC_CREDTIMEOUT	ユーザのクレデンシャルの有効期限が切れているかどうかを知らせる。
CS_SEC_SESSTIMEOUT	クライアントとサーバとの間のセッションの有効期限が切れているかどうかを知らせる。
CS_SEC_MUTUALAUTH	クライアント・アプリケーションはこのプロパティを設定して、サーバがそのサーバ自体であることをクライアントに対して認証することを要求する。
CS_SEC_DELEGATION	クライアント・アプリケーションはこのプロパティを設定して、ゲートウェイ・サーバがクライアントの委任クレデンシャル・トークンを使用してリモート・サーバに接続できるようにする。
CS_SEC_CREDENTIALS	ゲートウェイ・アプリケーションはこのプロパティを使用して、ゲートウェイのクライアントからリモート・サーバに委任クレデンシャル・トークンを転送する。

### ログインパスワード暗号化の FIPS-140-2 準拠

Open Client と Open Server のログイン・パスワードとリモート・パスワードの暗号化は、Sybase CSI (Common Security Infrastructure) によって実現されます。CSI 2.6 は、連邦情報処理標準 (FIPS: Federal Information Processing Standard) 140-2 に準拠しています。

FIPS 暗号化をサポートするため、まだ Certicom Security Builder を使用していないプラットフォームに *libsbgse2.so* (UNIX と Linux の各プラットフォーム) または *libsbgse2.dll* (Microsoft Windows プラットフォーム) という名前の Certicom Security Builder 共有ライブラリがインストールされます。また、*\$\$SYBASE/\$SYBASE\_OCS/lib3p* または *\$\$SYBASE/\$SYBASE\_OCS/lib3p64* にある *sybcsi* サブディレクトリは、削除されました。

ネットワーク認証は、すべてのセキュリティ・メカニズムでサポートされています。クレデンシャルとセッションのタイムアウトは、いくつかのセキュリティ・メカニズムではサポートされていますが、すべてのセキュリティ・メカニズムでサポートされているわけではありません。どのサービスがどのセキュリティ・メカニズムでサポートされているかについては、『Open Client/Server 設定ガイド』を参照してください。

『Open Client Library/C リファレンス・マニュアル』も参照してください。

## パケットごとのセキュリティ・サービス

環境によっては、分散アプリケーションが物理的に安全でないネットワークに対処しなければならない場合があります。たとえば権限のないグループが、物理回線にアナライザを接続したり、無線伝送を傍受したりする可能性があります。

これらの環境では、アプリケーションの保護と転送データの認証を使用して、安全なダイアログを保証してください。

次の表は、さまざまなパケットごとのサービスの使用方法を制御するプロパティを示します。

表 2-36 : データ認証プロパティ

プロパティ	説明
CS_SEC_ CONFIDENTIALITY	データ機密保持サービスを有効にする。 データ機密保持サービスはすべての転送データを暗号化して、不明なユーザが転送データを理解できないことを保証する。
CS_SEC_ INTEGRITY	データ整合性サービスを有効にする。 データ整合性サービスは、転送データを不正に変更しようとする動作が検出されることを保証する。
CS_SEC_ DATAORIGIN	データ・オリジン・スタンピング・サービスを有効にする。 データ・オリジン・スタンピング・サービスは、受信データが、そのクライアントまたはサーバから実際に送信されたことを保証する。
CS_SEC_ DETECTREPLAY	リプレイ検出サービスを有効にする。 リプレイ検出サービスは、受信した転送データを不明なユーザがリプレイ (再生) しようとする動作が検出されることを保証する。
CS_SEC_ DETECTSEQ	順序検証サービスを有効にする。 順序検証サービスは、送信された順序とは異なる順序で到達した転送を検出する。
CS_SEC_ CHANBIND	チャンネル・バインド・サービスを有効にする。 チャンネル・バインド・サービスは、クライアントのアドレスとサーバのアドレスの暗号化記述を使用して、それぞれの転送にマークを付ける。

**注意** この項で説明するサービスを使用するアプリケーションでは、クライアントとサーバ間のすべての通信でパケットごとのオーバーヘッドが生じます。アプリケーションのパフォーマンスよりもアプリケーションのセキュリティを重視する場合以外は、データ認証サービスを使用しないでください。

パケットごとのサービスすべてで、1つの接続で送信される TDS パケットごとに、次に示す処理のどちらかまたは両方が行われます。

- パケットの内容の暗号化
- パケット内容およびその他の必要な情報をコード化する、デジタル署名の計算



アプリケーションが複数のパケットごとのサービスを選択した場合、それぞれのオペレーションはパケットごとに一度だけ実行されます。たとえばアプリケーションがデータ機密性サービス、順序検証サービス、データ整合性サービス、チャンネル・バインド・サービスを選択した場合、それぞれのパケットは暗号化されて、パケットの内容、パケットの順序情報、ネットワーク・チャンネル識別子をコード化するデジタル署名が付加されます。

『Open Client Library/C リファレンス・マニュアル』を参照してください。

## SSL の概要

SSL は、クライアントからサーバ、およびサーバからサーバへワイヤまたはソケット・レベルで暗号化されたデータを送信する業界標準です。サーバとクライアントは何度か I/O を交換し、安全な暗号化セッションをネゴシエートして合意してから、SSL 接続が確立されます。これは、「SSL ハンドシェイク」と呼ばれています。

## SSL ハンドシェイク

クライアント・アプリケーションが接続を要求すると、SSL 対応サーバは証明書を提示し、ID を証明してから、データを送信します。基本的に、SSL ハンドシェイクは次の手順によって構成されています。

- クライアントはサーバに接続要求を送信します。要求には、クライアントがサポートしている SSL (または TLS: Transport Layer Security) オプションが含まれています。
- サーバは、証明書とサポートされている CipherSuite のリストを返します。これには、SSL/TLS サポート・オプション、キー交換で使用するアルゴリズム、デジタル署名が含まれます。
- クライアントとサーバがお互いに CipherSuite に合意すると、安全で暗号化されたセッションが確立されます。

SSL ハンドシェイクと SSL/TLS プロトコルの詳細については、Internet Engineering Task Force Web サイト (<http://www.ietf.org>) を参照してください。

## Open Client/Open Server の SSL

SSL には、いくつかのセキュリティ・レベルがあります。

- SSL 対応サーバへの接続を確立すると、サーバは接続対象のサーバであることを自己認証し、暗号化された SSL セッションが開始されてからデータが送信されます。
- SSL セッションが確立されると、ユーザ名とパスワードが暗号化された安全な接続によって送信されます。
- サーバ証明書のデジタル署名を比較して、サーバから受信したデータが転送中に変更されたかどうかを判断します。

## SSL フィルタ

SSL 対応 Adaptive Server Enterprise への接続を確立するとき、SSL セキュリティ・メカニズムは、`interfaces` ファイル (Windows では `sql.ini`) の `master` 行と `query` 行のフィルタとして設定されます。TCP/IP 接続の上層に位置する Open Client/Open Server プロトコル層として SSL を使用します。

SSL フィルタは、`interfaces` ファイル (Windows では `sql.ini`) の `SECHMECH` (security mechanism) 行で定義されている DCE や Kerberos などの他のセキュリティ・メカニズムとは異なります。`master` 行と `query` 行では、接続に使用されるセキュリティ・プロトコルを指定します。

たとえば、SSL を使用している UNIX マシンの一般的な `interfaces` ファイルは、次のようになります。

```
[SERVER]

query tcp ether hostname, port ssl
master tcp ether hostname, port ssl
```

SSL を使用している Windows 上の一般的な `sql.ini` ファイルは、次のようになります。

```
[SERVER]

query=TCP,hostname, port, ssl
master=TCP,hostname,port, ssl
```

`hostname` はクライアントが接続しているサーバの名前、`port` はホスト・マシンのポート番号です。`interfaces` ファイル内で SSL フィルタが指定されている `master` エントリまたは `query` エントリに接続するには、その接続で SSL プロトコルがサポートされている必要があります。サーバを、SSL 接続を受け入れ、他の接続によってプレーン・テキスト (非暗号化データ) を受け入れるように設定したり、他のセキュリティ・メカニズムを使用するように設定できます。

たとえば、SSL ベースの接続とプレーン・テキストの接続の両方をサポートする UNIX の Adaptive Server Enterprise の `interfaces` ファイルは、次のようになります。

```
SYBSRV1
master tcp ether hostname 2748 ssl
query tcp ether hostname 2748 ssl
master tcp ether hostname 2749
```

この例では、SSL セキュリティ・サービスがポート番号 2748 で定義されています。SYBSRV1 では、Adaptive Server Enterprise はクリア・テキストをポート番号 2749 で受信します。このポートには、セキュリティ・メカニズムやセキュリティ・フィルタは定義されていません。

### 証明書によるサーバの検証

Open Client/Open Server が SSL 対応サーバに接続する場合は、サーバは証明書ファイルが必要です。これには、サーバの証明書と暗号化プライベート・キーが含まれています。また、証明書は CA がデジタル署名したものでなければなりません。

既存のクライアント接続が確立されるのと同じように、Open Client アプリケーションは Adaptive Server Enterprise へのソケット接続を確立します。ネットワークのトランスポート層の接続コールがクライアント・サイドで完了し、受け入れコールがサーバ・サイドで完了すると、SSL ハンドシェイクが行われます。それから、ユーザのデータが送信されます。

SSL 対応サーバに正しく接続するには、次の手順に従ってください。

- クライアント・アプリケーションが接続要求を行った場合は、SSL 対応サーバは証明書を提出しなければなりません。
- クライアント・アプリケーションは、証明書に署名した CA を認識しなければなりません。「信頼された」CA すべてを含んだリストは、信頼されたルート・ファイルにあります。[「信頼されたルート・ファイル」\(205 ページ\)](#) を参照してください。
- SSL 対応サーバへの接続では、デフォルトの動作としてサーバ証明書の共通名と `interfaces` ファイルのサーバ名が比較されます。共有ディスク・クラスタ (SDC: Shared Disk Cluster) 環境では、クライアントはサーバ名または SDC インスタンス名とは無関係の SSL 証明書の共通名を指定できます。SDC 環境での共通名の検証については、[「SDC 環境での共通名の検証」\(204 ページ\)](#) を参照してください。

SSL 対応 Adaptive Server Enterprise への接続を確立すると、Adaptive Server Enterprise は起動時に次の場所からサーバ自体のコード化された証明書ファイルをロードします。

UNIX — `$$SYBASE/$SYBASE_ASE/certificates/servername.crt`

Windows — `%SYBASE%\¥%SYBASE_ASE%\¥certificates\¥servername.crt`

ここで、*servername* は、コマンド・ラインからサーバを起動したときに `-S` フラグで指定した Adaptive Server Enterprise の名前か、またはサーバの環境変数 `$DSLISTEN` で指定した Adaptive Server Enterprise の名前です。

ほかのタイプのサーバでは、別のロケーションに証明書を保管することがあります。サーバの証明書のロケーションの詳細については、ベンダ提供マニュアルを参照してください。

## SDC 環境での共通名の検証

Open Client と Open Server における SSL 検証のデフォルトの動作は、サーバ証明書での共通名を `ct_connect()` で指定されたサーバ名と比較することです。共有ディスク・クラスタ (SDC: Shared Disk Cluster) 環境では、クライアントはサーバ名または SDC インスタンス名とは無関係の SSL 証明書の共通名を指定できます。クライアントは、複数のサーバ・インスタンスを表すクラスタ名で SDC に接続することも、特定の 1 つのサーバ・インスタンスに接続することもできます。

クライアントはトランスポート・アドレスを使用して、証明書の検証で使用される共通名を指定できるため、Adaptive Server Enterprise の SSL 証明書の共通名がサーバ名やクラスタ名と異なってもかまいません。トランスポート・アドレスは、ディレクトリ・サービス (*interfaces* ファイル、LDAP、NT レジストリなど) のいずれか、または接続プロパティ `CS_SERVERADDR` で指定できます。

## UNIX での構文

UNIX での SSL 対応 Adaptive Server Enterprise およびクラスタのサーバ・エントリの構文を次に示します。

```
CLUSTERSSL
query tcp ether hostname1 5000 ssl="CN=name1"
query tcp ether hostname2 5000 ssl="CN=name2"
query tcp ether hostname3 5000 ssl="CN=name3"
query tcp ether hostname4 5000 ssl="CN=name4"

ASESSL1
master tcp ether hostname1 5000 ssl="CN=name1"
query tcp ether hostname1 5000 ssl="CN=name1"
```

```
ASESSL2
master tcp ether hostname2 5000 ssl="CN=name2"
query tcp ether hostname2 5000 ssl="CN=name2"
```

```
ASESSL3
master tcp ether hostname3 5000 ssl="CN=name3"
query tcp ether hostname3 5000 ssl="CN=name3"
```

```
ASESSL4
master tcp ether hostname1 5000 ssl="CN=name4"
query tcp ether hostname1 5000 ssl="CN=name4"
```

### Windows での構文

Windows での SSL 対応 Adaptive Server Enterprise およびクラスタのサーバ・エントリの構文を次に示します。

```
[CLUSTERSSSL]
query=tcp,hostname1,5000, ssl="CN=name1"
query=tcp,hostname2,5000, ssl="CN=name2"
query=tcp,hostname3,5000, ssl="CN=name3"
query=tcp,hostname4,5000, ssl="CN=name4"
```

```
[ASESSL1]
master=tcp,hostname1,5000, ssl="CN=name1"
query=tcp,hostname1,5000, ssl="CN=name1"
```

```
[ASESSL2]
master=tcp,hostname2,5000, ssl="CN=name2"
query=tcp,hostname2,5000, ssl="CN=name2"
```

```
[ASESSL3]
master=tcp,hostname3,5000, ssl="CN=name3"
query=tcp,hostname3,5000, ssl="CN=name3"
```

```
[ASESSL4]
master=tcp,hostname4,5000, ssl="CN=name4"
query=tcp,hostname4,5000, ssl="CN=name4"
```

### 信頼されたルート・ファイル

信頼された既知の CA のリストは、信頼されたルート・ファイルに保管されています。エンティティ(クライアント・アプリケーション、サーバ、ネットワーク・リソースなど)に既知の CA の証明書がある以外は、信頼されたルート・ファイルは証明書ファイルのフォーマットと同じです。システム・セキュリティ担当者が、標準 ASCII テキスト・エディタを使って認証局を追加したり、削除したりします。

Open Client/Open Server の信頼されたルート・ファイルは、次のロケーションにあります。

UNIX — `$$SYBASE/$SYBASE_OCS/config/trusted.txt`

Windows — `%SYBASE%\$SYBASE_OCS%\ini\trusted.txt`

現時点で認識されている CA は、Thawte、Entrust、Baltimore、VeriSign、RSA です。

デフォルトでは、Adaptive Server Enterprise はサーバ自身の信頼されたルート・ファイルを次のロケーションに保管します。

UNIX — `$$SYBASE/$SYBASE_ASE/certificates/servername.txt`

Windows — `%SYBASE%\$SYBASE_ASE%\certificates\servername.txt`

Open Client と Open Server の両方を使用すると、次のように信頼されたルート・ファイルを別のロケーションに設定できます。

- Open Client

```
ct_con_props (connection, CS_SET, CS_PROP_SSL_CA,
              "$SYBASE/config/trusted.txt", CS_NULLTERM, NULL);
```

`$$SYBASE` はインストール・ディレクトリです。 `ct_config()` を使ってコンテキスト・レベルに、または `ct_con_props()` を使って接続レベルに `CS_PROP_SSL_CA` を設定できます。

- Open Server:

```
srv_props (context, CS_SET, SRV_S_CERT_AUTH,
           "$SYBASE/config/trusted.txt", CS_NULLTERM, NULL);
```

`$$SYBASE` はインストール・ディレクトリです。

`bcp` ユーティリティと `isql` ユーティリティでも、別の場所にある信頼されたルート・ファイルを指定できます。新しいパラメータ `-x` が構文に追加されており、このパラメータを使用して `trusted.txt` ファイルの場所を指定します。

SSL およびパブリック・キー暗号法については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

## Open Server でのセキュリティ・サービスの機能

セキュリティ・サービスを開始するには、クライアントはダイアログを確立するときに、セキュリティ・メカニズムにマップされる「オブジェクト識別子」をサーバに送信します。サーバは、そのオブジェクト識別子をセキュリティ・メカニズムのローカル名にマップします。要求されたセキュリティ・メカニズムをサーバがサポートしていない場合や、セキュリティ・セッションをまったくサポートしていない場合、ダイアログの要求は失敗して、Open Server はエラーを返します。

オブジェクト識別子を使用すると、セキュリティ・メカニズムのローカル名はクライアントとサーバで異なってもかまいません。この場合、システム管理者とアプリケーション・プログラマは、セキュリティ・メカニズムに対して独自の別々のローカル命名規則を開発できます。「オブジェクト識別子」(210 ページ) を参照してください。

Server-Library を使用すると、クレデンシャルを取得するときに使用されるプリンシパル名を指定できます。この「プリンシパル名」は、セキュリティ・サービス・プロバイダが Open Server アプリケーションを認識している名前です。アプリケーションのプリンシパル名を設定するには、SRV\_S\_SEC\_PRINCIPAL サーバ・プロパティを指定して `srv_props` 関数を実行します。

プリンシパル名が設定されていない場合のデフォルト値は Open Server アプリケーションのネットワーク名ですが、このネットワーク名は一般に、`srv_init` で指定されます。

Open Server はクライアントとのセキュリティ・セッションを確立するときに、クレデンシャルを使用します。

クライアントのログイン名はセキュリティ・セッションから得られます。したがって、ログイン・レコード内にどのような名前が指定されていても無視されます。

セキュリティ・サービスを使用する場合のクライアントの役割については、『Open Client Library/C リファレンス・マニュアル』を参照してください。

### セキュリティ・サービスを使用する Client/Server ダイアログに必要な手順

クライアントがセキュリティ・サービスを使用するダイアログを開始する場合、Open Server は次の手順に従います。

- 1 クライアントとのトランスポート接続を確立します。

- 2 クライアントのログイン・レコードと内部が隠されたセキュリティ・トークンを受信して、内部が隠された必要なトークンでクライアントに応答します。
- 3 セキュリティ・メッセージ・ハンドシェイクが成功したときに、セキュリティ・セッションを確立します。

Open Server アプリケーションがクライアントから情報を受信した場合は、次の手順に従います。

- 1 クライアントから受信した応答に関連付けられているセキュリティ・メッセージ(暗号化シグニチャなど)がある場合は、処理します(暗号化シグニチャによって、メッセージの整合性が確保されます)。
- 2 セキュリティ・セッションでサポートされているセキュリティ・サービスに基づいて、適切なルーチン(シグニチャを検証するルーチンなど)を呼び出します。
- 3 TDS (Tabular Data Stream) に対し、通常の処理を実行します。

Open Server は、次の手順に従ってクライアントに応答を送信します。

- 1 クレデンシャルまたはセキュリティ・セッションの有効期限が切れていないかどうかをチェックします。有効期限切れが検出された場合、Open Server はエラーの処理を実行します。
- 2 このダイアログでサポートされているセキュリティ・サービスに基づいて、適切なルーチン(応答に対して暗号化シグニチャを生成するルーチンなど)を呼び出します。
- 3 要求された TDS を生成して、パケットごとのセキュリティ・サービスを識別します。

セキュリティ・セッションは、対応するクライアント・ダイアログが終了したときに終了します。セキュリティ・セッションは通常のクライアント・ログアウトで終了する場合もあれば、エラー条件が発生したために終了する場合があります。

## Open Server アプリケーションでのセキュリティ・メカニズムの使用

この項では、Open Server アプリケーションでサード・パーティのセキュリティ・メカニズムを使用するために行う必要がある変更について説明します。これらの変更には、次のようなエントリを追加する作業も含まれます。



- *libtcl.cfg* ファイル内で、セキュリティ・メカニズムからドライバへの各マッピングのエントリを追加する作業。
- グローバル・オブジェクト識別ファイル *objectid.dat* 内で、グローバルにユニークなオブジェクト識別子に各セキュリティ・メカニズムのローカル名をマップするエントリを追加する作業。
- サード・パーティのセキュリティ・メカニズムを使用する各サーバの *interfaces* ファイル内で、サーバがサポートしているすべてのセキュリティ・メカニズムを指定するエントリを追加する作業。

## セキュリティ・ドライバ

Sybase は「セキュリティ・ドライバ」を提供しています。このセキュリティ・ドライバを使用すると、Client-Library アプリケーションと Server-Library アプリケーションは、インストールされたネットワーク・セキュリティ・システムを活用できます。Client-Library と Server-Library には、安全化されているアプリケーションを実装するための汎用インタフェースが備わっています。この汎用インタフェースは、それぞれの Sybase セキュリティ・ドライバによってセキュリティ・プロバイダのインタフェースにマップされます。

セキュリティ・ドライバは動的にロードでき、1 つまたは複数のセキュリティ・メカニズムをサポートしています。

現在サポートされている各セキュリティ・プロバイダのドライバは、次のとおりです。

- *libsybsdce*  
DCE セキュリティ・サービスの場合
- *libsybsmssp*  
Microsoft NT SSPI の場合

## *libtcl.cfg* 設定ファイル

*libtcl.cfg* 設定ファイルは、セキュリティ・メカニズムのローカル名を、そのメカニズムをサポートするのに必要なセキュリティ・ドライバにマップします。*libtcl.cfg* ファイルは、`$SYBASE/$SYBASE_OCS/config` ディレクトリ、またはコンテキスト・プロパティ `CS_LIBTCL_CFG` により指定されるパスにあります。このファイルの正確なロケーションについては、使用しているプラットフォーム用の『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

*libtcl.cfg* ファイル内には、それぞれのセキュリティ・ドライバのエントリが指定されている必要があります。それぞれのドライバは1つのセキュリティ・メカニズムをサポートしている場合もあれば、複数のセキュリティ・メカニズムをサポートしている場合もあります。ドライバが複数のセキュリティ・メカニズムをサポートしている場合は、*libtcl.cfg* ファイル内にそれぞれのセキュリティ・メカニズムのエントリが指定されている必要があります。

このファイルのフォーマットは次のとおりです。

[SECURITY]

```
local-name-of-security-mechanism = path-to-the-driver init-string
```

各パラメータの意味は次のとおりです。

- *path-to-the-driver* — オブジェクト・ファイルへの完全に修飾されたパス名です。
- *init-string* — は各ドライバに応じて異なる引数リストであり、次の一般的フォームで表記されます。 *token = value, token = value, ...*

以下は UNIX プラットフォームで指定する場合の例です。

[SECURITY]

```
csfkrb5=libsybskrb.so secbase=@MYREALM libgss=/krb5/lib/libgss.so
```

*libtcl.cfg* ファイル内の最初のエントリは、デフォルトのセキュリティ・メカニズムです。アプリケーションがセキュリティ・サービスを要求する場合に、**Open Server** はデフォルトのセキュリティ・メカニズムを使用しますが、セキュリティ・メカニズムの設定は行いません。

詳細については、使用しているプラットフォームの『**Open Client/Server 設定ガイド**』を参照してください。

## オブジェクト識別子

それぞれのセキュリティ・メカニズムには、対応するオブジェクト識別子があります。グローバルにユニークなオブジェクト識別子は、グローバル・オブジェクト識別ファイル *Objectid.dat* 内で、1つのセキュリティ・メカニズムのローカル名にマップされます。これによって、柔軟で一貫性のある方法で、クライアントとサーバとの間でセキュリティ・メカニズム名を通信できます。*Objectid.dat* ファイルは、*\$SYBASE/config* ディレクトリにあります。

グローバル識別ファイルのフォーマットは次のとおりです。

```
[Object Class]
Object_Identifier Object_Name_List
```

セキュリティ・メカニズムのエントリは次のようになります。

*Object Class* — "secmech" です。

*Object\_Identifier* — ドットで区切られた負でない一連の整数値です。オブジェクト識別子は、国際的な標準化組織である CCITT と ISO で定義された用語規定に基づいています。DCE セキュリティ・ドライバの *sybase* ルートからのオブジェクト識別子の例は、897.4.6.1 です。

*Object\_Name\_List* — ローカル・セキュリティ・メカニズム名をカンマで区切ったリストです。

次に例を示します。

```
[secmech]
1.3.6.1.4.1.897.4.6.3 = NTLM
```

## interfaces ファイルへの変更

*interfaces* ファイルのフォーマットは、サーバがサポートしているセキュリティ・メカニズムを指定できるように拡張されています。フォーマットは次のとおりです。

```
SERVERNAME
    query tcp sun-ether joyce 2901
    master tcp sun-ether joyce 2901
    secmech mechanism1, mechanism2, ..., mechanismN
```

*secmech* 識別子は、サーバがサポートしているすべてのセキュリティ・メカニズムのリストです。また、次の条件が適用されます。

- この行はオプションであり、サーバが Sybase 独自のセキュリティ・メカニズムを使用していない場合にだけ使用されます。
- *interfaces* ファイル内にサーバの *secmech* エントリがない場合、そのサーバは *libtcl.cfg* ファイルの *secmech* エントリで指定されているすべてのセキュリティ・メカニズムをサポートしています。
- *interfaces* ファイル内にサーバの *secmech* エントリがあるが、セキュリティ・メカニズムが指定されていない場合、そのサーバはどのようなセキュリティ・メカニズムもサポートしていません。

*mechanism1, mechanism2, ... mechanismN* は、サーバがサポートしているセキュリティ・メカニズムのオブジェクト識別子です。カンマをセパレータとして使用して複数のセキュリティ・メカニズムを指定できます。「オブジェクト識別子」(210 ページ)を参照してください。

## interfaces ファイルへの変更 : SSL フィルタ

SSL フィルタは、interfaces ファイル (Windows では *sql.ini*) の SECMECH (security mechanism) 行で定義されている DCE や Kerberos などの他のセキュリティ・メカニズムとは異なります。master 行と query 行では、接続に使用されるセキュリティ・プロトコルを指定します。

たとえば、SSL を使用している UNIX マシンの一般的な interfaces ファイルは、次のようになります。

```
[SERVER]
query tcp ether hostname port ssl
master tcp ether hostname port ssl
```

SSL を使用している Windows 上の一般的な *sql.ini* ファイルは、次のようになります。

```
[SERVER]

query=TCP,hostname, port, ssl
master=TCP,hostname,port, ssl
```

*hostname* はクライアントが接続しているサーバの名前、*port* はホスト・マシンのポート番号です。interfaces ファイル内で SSL フィルタが指定されている master エントリまたは query エントリに接続するには、その接続で SSL プロトコルがサポートされている必要があります。サーバを、SSL 接続を受け入れ、他の接続によってプレーン・テキスト (非暗号化データ) を受け入れるように設定したり、他のセキュリティ・メカニズムを使用するように設定できます。

## アクティブなセキュリティ・サービスの決定

クライアント・サーバ間ダイアログでどのセキュリティ・サービスがアクティブであるかを調べるには、*srv\_thread\_props* を使用して、次のスレッド・プロパティの値を取得します。

- SRV\_T\_SEC\_CHANBIND
- SRV\_T\_SEC\_CONFIDENTIALITY
- SRV\_T\_SEC\_DATAORIGIN
- SRV\_T\_SEC\_DELEGATION
- SRV\_T\_SEC\_DETECTREPLAY
- SRV\_T\_SEC\_DETECTSEQ
- SRV\_T\_SEC\_INTEGRITY

- SRV\_T\_SEC\_MUTUALAUTH
- SRV\_T\_SEC\_NETWORKAUTH

各スレッド・プロパティの説明については、表 2-28 (169 ページ) を参照してください。

## Open Server アプリケーションでのセキュリティ・サービスの使用例

この項では、さまざまな Open Server アプリケーション設定でセキュリティ・サービスを使用する方法について説明します。具体的には、次の場合について説明します。

- セキュリティ・セッションを使用する単純な Open Server アプリケーション
- 個別のセキュリティ・セッションを使用するゲートウェイ Open Server アプリケーション
- 委任を使用して個別のセキュリティ・セッションを使用するゲートウェイ Open Server アプリケーション
- ダイレクト・セキュリティ・セッションを使用するフル・パスルー・ゲートウェイ Open Server アプリケーション

### セキュリティ・セッションを使用する単純なアプリケーション

最も単純な設定では、クライアントはセキュリティ・メカニズムによって提供される認証サービスを使用して、ダイアログを確立します。Open Server はログイン・ネゴシエーションを実行してから、接続ハンドラを呼び出します。接続ハンドラによって `srv_senddone(SRV_DONE_FINAL)` が発行されると、Open Server はステータスが "success" のログイン確認をクライアントに送信します。

この設定では、接続ハンドラをインストールする必要はありません。デフォルトの接続ハンドラで十分です。それでも接続ハンドラをインストールする場合は、次の例で示すように少なくとも `srv_senddone(SRV_DONE_FINAL)` だけは送信してください。

```
CS_RETCODE CS_PUBLIC connect_handler(spp)
SRV_PROC      *spp;
{
    .....
    /*
```

```

** You do not need to test this srv_senddone's return value
** since Open Server will kill this thread if this call fails.
*/
(CS_VOID)srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED,
                      (CS_INT)0);
return (CS_SUCCEED);
}

```

## 個別のセキュリティ・セッションを使用するゲートウェイ・アプリケーション

以下に示す例では、Open Server アプリケーションはクライアントと別のサーバの間のゲートウェイとして動作します。クライアントとゲートウェイ・アプリケーションの間のセキュリティ・セッションを確立するのに使用されるネットワーク ID は、ゲートウェイとリモート・サーバの間のセキュリティ・セッションを確立するのに使用されるネットワーク ID とは異なる場合があります。

ゲートウェイ・アプリケーションは最後のログイン確認を保留して、そのクライアントとのログイン・セキュリティ・ネゴシエーションを完了してから、接続ハンドラを呼び出します。接続ハンドラは Client-Library 呼び出しを使用して、リモート・サーバへのセキュリティ・セッション・ベースのログインを開始してから、`srv_senddone(SRV_DONE_FINAL)` をクライアントに送信してそのログインを完了する必要があります。次は、接続ハンドラの例です。

```

CS_RETCODE CS_PUBLIC connect_handler(spp)
SRV_PROC      *spp;
{
    CS_CONNECTION *conn;    /* the connection handle */
    CS_BOOL      trueval = CS_TRUE;
    CS_INT      outlen;

    .....

    allocate and set user data in spp...

    .....

    /* Allocate a connection handle */
    if (ct_con_alloc(Context, &(userdata->conn)) == CS_FAIL)
    {
        clean up and report error...
    }
    return(CS_FAIL);
}

.....

conn = userdata->conn;

```

```
/*
** Initiate security session based login with the remote
** server. The user name used here may be the same as the
** client user name or different
*/
if (ct_con_props(conn, CS_SET, CS_USERNAME,
    (CS_VOID*)Username, STRLEN(Username), (CS_INT*)NULL)
    == CS_FAIL)
{
    handle failure...
}

/*
** Set the desired security mechanism(s) or use the default
** security mechanism.
*/
if (ct_con_props(conn, CS_SET, CS_SEC_MECHANISM,
    (CS_VOID*)Mechanismname, STRLEN(Mechanismname),
    (CS_INT*)NULL) == CS_FAIL)
{
    handle failure...
}

/* Set the security service-network authentication */
if (ct_con_props(conn, CS_SET, CS_SEC_NETWORKAUTH,
    (CS_VOID*)&trueval, CS_SIZEOF(CS_BOOL), (CS_INT*)NULL)
    == CS_FAIL)
{
    handle failure...
}

set other security services if required
get and set the user's application name, response capabilities
set the locale and other login properties
/* Attempt a connection to the remote server */
if (ct_connect(conn, Servername, CS_NULLTERM) == CS_FAIL)
{
    cleanup...
    return(CS_FAIL);
}

get and set the REQUEST capabilities
get and set the RESPONSE capabilities
.....
/*
** You do not need to test this srv_senddone's return value
** since Open Server will kill this thread if this call fails.
*/
```

```
(CS_VOID)srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED,
                      (CS_INT)0);
return (CS_SUCCEED);
}
```

## 委任を使用して個別のセキュリティ・セッションを使用するゲートウェイ

Open Server アプリケーションはクライアントと別のサーバの間のゲートウェイとしても動作しますが、ゲートウェイ・アプリケーションはリモート・サーバとのセキュリティ・セッションを確立するときに、委任クライアント・クレデンシャルを使用します。クライアントは、そのクライアント自体のクレデンシャルだけを委任できます。

一度セキュリティ・セッションが確立されると Open Server アプリケーションが委任クレデンシャルを得ることができるように、クライアントは CS\_SEC\_DELEGATION サービスを要求する必要があります。

クライアントとゲートウェイ Open Server アプリケーションとの間のセキュリティ・セッションは、最後のログイン確認を除いて、「[セキュリティ・セッションを使用する単純なアプリケーション](#)」(213 ページ)と同様に確立されます。

接続ハンドラ内では、ゲートウェイ・アプリケーションは次の手順に従います。

- 1 `srv_thread_props(CS_GET, SRV_T_SEC_DELEGCRED)` を使用して、委任クレデンシャルを取得します。
- 2 `ct_con_props(CS_SET, CS_SEC_CREDENTIALS)` を使用して、リモート・サーバへの接続に使用するための委任クレデンシャルを Client-Library 接続構造体内に設定します。
- 3 `ct_connect` ルーチンを使用して、リモート・サーバに接続しようとしています。
- 4 `srv_senddone(SRV_DONE_FINAL)` を送信して、クライアントのログインを確認します。

次は、接続ハンドラの例です。

```
CS_RETCODE CS_PUBLIC connect_handler(spp)
SRV_PROC      *spp;
{
    CS_CONNECTION *conn;      /* Connection handle */
    CS_VOID      *creds;      /* security credentials */
    CS_BOOL      trueval = CS_TRUE;
    CS_BOOL      boolval;
    CS_CHAR      mechanismname[MAX_NAMESIZE];
```



```
CS_CHAR      username[MAX_NAMESIZE];
CS_INT       outlen;
.....
allocate and set user data in spp...
.....
/* Allocate a connection handle for the connection attempt.*/
if (ct_con_alloc(Context, &(userdata->conn)) == CS_FAIL)
{
    return(CS_FAIL);
}
.....
conn = userdata->conn;
/*
** Initiate security session based login to the target server
*/
/* Retrieve the client user name */
if (srv_thread_props(spp, CS_GET, SRV_T_USER,
    (CS_VOID *)username, MAX_NAMESIZE, &outlen) == CS_FAIL)
{
    handle failure...
}
/*
** Set the client's security principal name to connect to the
** target server
*/
if (ct_con_props(conn, CS_SET, CS_USERNAME,
    (CS_VOID *)username, outlen, (CS_INT *)NULL) == CS_FAIL)
{
    handle failure...
}
/* Retrieve and set the security mechanism */
if (srv_thread_props(spp, CS_GET, SRV_T_SEC_MECHANISM,
    (CS_VOID *)mechanismname, MAX_NAMESIZE, &outlen)
    == CS_FAIL)
{
    handle failure...
}
if (ct_con_props(conn, CS_SET, CS_SEC_MECHANISM,
    (CS_VOID *)mechanismname, outlen, (CS_INT *)NULL)
    == CS_FAIL)
{
    handle failure...
}
/*
** Set security service-network authentication. Alternatively
** retrieve services from the current thread and set it.
```

```

*/
if (ct_con_props(conn, CS_SET, CS_SEC_NETWORKAUTH,
    (CS_VOID*)&trueval, CS_SIZEOF(CS_BOOL), (CS_INT*)NULL)
    == CS_FAIL)
{
    handle failure...
}
set other security services if needed...
/* Ensure that the client enabled security delegation */
if (srv_thread_props(spp, CS_GET, SRV_T_SEC_DELEGATION,
    (CS_VOID *)&boolval, CS_SIZEOF(CS_BOOL), (CS_INT *)NULL)
    == CS_FAIL)
{
    handle failure...
}
if (boolval != CS_TRUE)
{
    /* delegation not handled on this dialog */
    handle failure...
}
/* Retrieve the delegated credentials */
if (srv_thread_props(spp, CS_GET, SRV_T_SEC_DELEGCRED,
    (CS_VOID *)&creds, CS_SIZEOF(CS_VOID*), (CS_INT *)NULL)
    == CS_FAIL)
{
    handle failure...
}
/*
** Set the delegated credentials to authenticate to the target
** server.
*/
if (ct_con_props(conn, CS_SET, CS_SEC_CREDENTIALS,
    (CS_VOID *)&creds, CS_SIZEOF(CS_VOID*), (CS_INT *)NULL)
    == CS_FAIL)
{
    handle failure...
}
get and set the user's application name and response
capabilities...
set the locale and other properties...
/* Attempt a connection to the remote server */
if (ct_connect(conn, Servername, CS_NULLTERM) == CS_FAIL)
{
    handle failure...
}
Get and set the REQUEST capabilities...

```

```

Get and set the RESPONSE capabilities...
.....
/*
** You do not need to test this srv_senddone' s return value
** since Open Server will kill this thread if this call fails.
*/
(CS_VOID)srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED,
                      (CS_INT)0);
return (CS_SUCCEED);
}

```

## ダイレクト・セキュリティ・セッションでのフル・パススルー・ゲートウェイ

クライアントがセキュリティ・セッションを確立するには、リモート・サーバが必要です。クライアントとリモート・サーバの間のどのような中間サーバでも、パケットごとのセキュリティ・サービスは実行されません。クライアントが機密保持を要求する場合、ゲートウェイはメッセージ・パケットから TDS トークンを取得できません。この設定では、受信パケットを復号化するのに使用されるサービスや、転送前に受信パケットを再暗号化するのに使用されるサービスなどのパケットごとのサービスはゲートウェイ内では実行されないで、オーバーヘッドが減ります。

一連の「転送サーバ」を構成する複数のゲートウェイ手段が介在する場合があります。この場合、それぞれの転送サーバは同じセキュリティ・メカニズムをサポートしている必要があります。

ダイレクト・セキュリティ・セッションを設定するには、**Open Server** ゲートウェイ・アプリケーションの接続ハンドラで次の手順に従います。

- 1 `srv_getloginfo` ルーチンを使用して、クライアント・スレッドからログイン情報を得ます。
- 2 `ct_setloginfo` ルーチンを使用して、リモート・サーバへの接続に使用される接続構造体内に、この情報を設定します。
- 3 次のコマンドを使用して、セキュリティ・セッション・コールバックをインストールします。

```
ct_callback(conn, CS_SET, CS_SECSESSION_CB, secsession_cb)
```

リモート・サーバへの接続が確立されると、コールバックは、リモート・サーバとゲートウェイのクライアントとの間で必要なハンドシェイクの仲介者として動作します。

コールバックの内容については、「[セキュリティ・セッション・コールバック](#)」(222 ページ)を参照してください。

コールバックの詳細については、『Open Client Library/C リファレンス・マニュアル』を参照してください。

- 4 `ct_connect` ルーチン呼び出して、リモート・サーバに接続します。この呼び出しはクライアントとリモート・サーバの間のネゴシエーションを開始して、セキュリティ・セッションを確立します。`ct_connect` ルーチンが `CS_SUCCEED` を返した場合は、セキュリティ・セッションの確立が成功したことを示します。
- 5 `srv_senddone(SRV_DONE_FINAL)` を使用して、ログインが完了したことをクライアントに通知します。

### 接続ハンドラの例

```
CS_RETCODE CS_PUBLIC connect_handler(spp)
SRV_PROC      *spp;
{
    CS_CONNECTION *conn;      /* connection handle */
    CS_VOID      *creds;      /* security credentials */
    CS_LOGININFO *logininfo; /* login information */
    CS_BOOL boolval;

    .....
    allocate and set user data in spp...
    /* Allocate a connection handle for the connection attempt.*/
    if (ct_con_alloc(Context, &(userdata->conn)) == CS_FAIL)
    {
        handle failure...
    }
    .....
    conn = userdata->conn;
    /*
    ** Save the pointer to thread control structure in the
    ** connection handle
    */
    if (ct_con_props(conn, CS_SET, CS_USERDATA, &spp,
        CS_SIZEOF(spp), (CS_INT *)NULL) == CS_FAIL)
    {
        handle failure...
    }
    /* Verify that security based login is requested */
    if (srv_thread_props(spp, CS_GET, SRV_T_SEC_NETWORKAUTH,
        (CS_VOID *)&boolval, CS_SIZEOF(CS_BOOL), (CS_INT *)NULL)
        == CS_FAIL)
    {
        handle failure...
    }
}
```

```
if (boolval != CS_TRUE)
{
    handle the client request that does not use security
    session based login
    .....
    return (CS_SUCCEED);
}

/* Get and set the login information */
if (srv_getloginfo(spp, &loginfo) == CS_FAIL)
{
    handle failure...
}
if (ct_setloginfo(conn, loginfo) == CS_FAIL)
{
    handle failure...
}
/* Install a security session callback for this connection */
if (ct_callback((CS_CONTEXT *)NULL, conn, CS_SET,
    CS_SECSESSION_CB, (CS_VOID *)secsession_cb) == CS_FAIL)
{
    handle failure...
}
/* Attempt a connection to the remote server */
if (ct_connect(conn, Servername, CS_NULLTERM) == CS_FAIL)
{
    handle failure...
}
/* Get and set the login information */
if (ct_getloginfo(conn, &loginfo) == CS_FAIL)
{
    handle failure...
}
if (srv_setloginfo(spp, loginfo) == CS_FAIL)
{
    handle failure...
}
.....
/*
** You do not need to test this srv_senddone's return value
** since Open Server will kill this thread if this call fails.
*/
(CS_VOID)srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED,
    (CS_INT)0);
return (CS_SUCCEED);
}
```

## セキュリティ・セッション・コールバック

セキュリティ・セッション・コールバック・ルーチンは、ターゲット・サーバ(またはそのゲートウェイの次の中間ゲートウェイ)とゲートウェイのクライアント・アプリケーションとの間でセキュリティ・トークンを交換して、クライアントとリモート・サーバとの間のダイレクト・セキュリティ・セッションを確立します。このコールバック手順は、異なるパラメータを使用することを除けば、チャレンジ/応答コールバックと似ています。

ゲートウェイが `ct_connect` ルーチンを呼び出すと、リモート・サーバはセキュリティ・セッション情報が入っている1つまたは複数のメッセージを発行します。それぞれのセキュリティ・メッセージに対して、**Client-Library** はリモート・サーバから送信されたメッセージ・パラメータを使用して、コールバックを開始します。

コールバック・ルーチンは次の手順に従う必要があります。

- 1 リモート・サーバのメッセージからパラメータを取得します。
- 2 次のルーチンを使用して、クライアントにパラメータを送信します。
  - `srv_negotiate(..., CS_SET, SRV_NEG_SECSSESSION)`
  - `srv_descfmt(..., CS_SET, SRV_NEGDATA, ...)`
  - `srv_bind(..., CS_SET, ...)`
  - `srv_xferdata(..., CS_SET, ...)`
- 3 クライアントに `srv_senddone(SRV_DONE_FINAL)` を送信します。
- 4 `srv_negotiate(CS_GET, SRV_NEG_SECSSESSION)` を使用して、クライアントからの応答を待ちます。
- 5 クライアントが応答すると、コールバック・ルーチンは、次の関数を使用して、対応するセッション・データをクライアントから出力バッファにコピーし、リモート・サーバに送信します。
  - `srv_descfmt(CS_GET)`
  - `srv_bind(CS_GET)`
  - `srv_xferdata(CS_GET)`
- 6 リモート・サーバがまた別のセキュリティ・メッセージを送信する場合は、この処理を繰り返します。

セキュリティ・セッション・コールバックの定義については、『**Open Client Client-Library/C** リファレンス・マニュアル』を参照してください。

**Client-Library セキュリティ・セッション・コールバック・ルーチンの例**

```

CS_RETCODE CS_PUBLIC secsession_cb(conn, innumparams, infmt,
    inbuf, outnumparams, outfmt, outbuf, outlen)
CS_CONNECTION *conn;
CS_INT innumparams;
CS_DATAFMT *infmt;
CS_BYTE **inbuf;
CS_INT *outnumparams;
CS_DATAFMT *outfmt;
CS_BYTE **outbuf;
CS_INT *outlen;
{
    SRV_PROC *spp; /* The SRVPROC structure associated with the
        ** client connection */
    CS_INT i;

    /* Get the previously saved spp for the client */
    if (ct_con_props(conn, CS_GET, CS_USERDATA, &spp,
        CS_SIZEOF(spp), (CS_INT *)NULL) != CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    /*
    ** Use srv_negotiate to tell the client to expect a security
    ** token
    */
    if (srv_negotiate(spp, CS_SET, SRV_NEG_SECSSESSION)
        != CS_SUCCEED)
    {
        return(CS_FAIL);
    }

    /* Describe and send the security token */
    for (i = 0; i < innumparams; i++)
    {
        if (srv_descfmt(spp, CS_SET, SRV_NEGDATA, i + 1, &infmt[i]
            != CS_SUCCEED)
        {
            return(CS_FAIL);
        }

        if (srv_bind(spp, CS_SET, SRV_NEGDATA, i + 1, &infmt[i],
            inbuf[i], &(infmt[i]->maxlength), (CS_SMALLINT *)NULL)
            != CS_SUCCEED)
        {
            return(CS_FAIL);
        }
    }
}

```

```
    }
    if (srv_xferdata(spp, CS_SET, SRV_NEGDATA) != CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
    /* Complete this portion of the exchange */
    if (srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED, 0)
        != CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
    /* Wait until the client responds */
    if (srv_negotiate(spp, CS_GET, SRV_NEG_SECSSESSION)
        != CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
    /* Get the number of parameters in the client's response */
    if (srv_numparams(spp, outnumparams) != CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
    /* Read in the client's response */
    for (i = 0; i < (*outnumparams); i++)
    {
        srv_bzero(&outfmt[i], sizeof(CS_DATAFMT));
        if (srv_descfmt(spp, CS_GET, SRV_NEGDATA, i + 1, &outfmt[i]
            != CS_SUCCEEDED)
            {
                return(CS_FAIL);
            }
        if (srv_bind(spp, CS_GET, SRV_NEGDATA, i + 1, &outfmt[i],
            outbuf[i], &outlen[i], (CS_SMALLINT *)NULL)
            != CS_SUCCEEDED)
            {
                return(CS_FAIL);
            }
    }
    if (srv_xferdata(spp, CS_GET, SRV_NEGDATA) != CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
}
```



```
}  
/* Return success */  
return (CS_SUCCEED);  
}
```

## text と image

`text` と `image` Adaptive Server Enterprise データ型には、大きな `text` または `image` の値が入ります。`text` データ型は、印刷可能文字が最大 2,147,483,647 バイト入ります。`image` データ型は、バイナリ・データが最大 2,147,483,647 バイト入ります。

`text` および `image` 値は、非常に大きくなる可能性があるため、実際にはデータベース・テーブルには保存されません。代わりに `text` 値または `image` 値へのポインタがテーブルに格納されます。このポインタを「テキスト・ポインタ」と呼びます。

競合するクライアント・アプリケーションが、他方が行ったデータベースへの変更を上書きしないように、各 `text` または `image` カラムにタイムスタンプが関連付けられています。このタイムスタンプを、「テキスト・タイムスタンプ」と呼びます。

## text および image データの処理

クライアントは、`writetext` ストリームと呼ばれる無区別のデータ・ストリームの形で `text` および `image` データを送ります。パラメータに区別されていないため、Open Server アプリケーションでは、受信パラメータ・データの処理に通常使用するルーチンを使用できません。これらのルーチンとは、`srv_descfmt`、`srv_bind`、`srv_xferdata` です。そこで、`text` および `image` ルーチンの特殊なセットを使用する必要があります。

Open Server アプリケーションは、リターン・ローにどれだけのカラムが含まれているかによって、2つの方法のどちらかを使って text または image データをクライアントに返すことができます。リターン・ローにカラムが1つだけ含まれていて、それに text データまたは image データが含まれている場合、そのカラムは無区別データ・ストリームとして扱われて通常とは異なる処理になります。ただし、リターン・ローに text カラムまたは image カラム以外に別のカラムが含まれている場合、text データまたは image データは記述/バインド/転送方法を使って処理されます。両方に、共通の手順があることに注意してください。

記述/バインド/転送の3つの呼び出しの詳細については、「[パラメータとロー・データの処理](#)」(152 ページ)を参照してください。

## クライアントからのデータの取得

writetext ストリームは、SRV\_BULK イベントをトリガします。クライアントから取得された text および image データはバルク・データとして扱われるので、Open Server アプリケーションは、入ってくる text および image データを、バルク・ハンドラの中から処理します。『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

アプリケーションは、受信 text および image データを次の2段階で処理します。

- 1 `srv_text_info` ルーチンは、text データまたは image データの記述を取得し、その情報を `CS_IODESC` 構造体に格納します。この呼び出しは、さまざまな情報を返しますが、その中で最も重要なのが、データの総合の長さです。この長さに基づいて、アプリケーションはデータを一度に取得するかセクションに分けて取得するかを決定でき、またデータを保存するバッファの大きさも決めることができます。`srv_text_info` は、`cmd` 引数を `CS_GET` に設定して呼び出します。
- 2 `srv_get_text` ルーチンは、指定されたセクション・サイズでクライアントから実際にデータを取得し、指定されたバッファに保存します。

`srv_get_text` の呼び出しの前に、必ず `srv_text_info` を呼び出すことに注意してください。`srv_get_text` ルーチンは、すべての text がクライアントから読み込まれるまで、呼び出し続けなければなりません。

## コマンドの抑制

text または image カラムの更新処理を単純化しパフォーマンスを向上させるために、クライアントは SQL コマンド (update または writetext) の生成を抑制し、サーバのバルク・ハンドラに直接データを送信することもできます。そのためには、クライアントは *type* パラメータを `CS_SEND_DATA_NOCMD` に設定し、`ct_command` ルーチン呼び出すことによってデータ送信コマンドを開始しなければなりません。その後、クライアント・アプリケーションは、データ送信コマンドを使用してサーバのバルク・ハンドラに text データのみまたは image データのみを送信できます。サーバでバルク・イベントが発生すると、送信する合計バイト数を示す 4 バイトのフィールドに続き、text または image データが送信されます。バルク・ハンドラは `srv_text_info` を使用して予想される合計バイト数を読み取り、`srv_get_data` を使用してデータを読み取ります。

サーバはストアド・プロシージャ `sp_mda` を定義して、SQL コマンドを使用せずに text データまたは image データのみを送信する `ct_send_data` ルーチンをそれがサポートするかどうかを指定しなければなりません。サーバの `sp_mda` プロシージャは、`ct_connect` ルーチンの呼び出し前に、クライアント・アプリケーションで `ct_con_props(CS_SENDDATA_NOCMD)` などの `CS_SENDDATA_NOCMD` 接続プロパティが設定されている場合のみ呼び出されます。`CS_SENDDATA_NOCMD` が設定されている場合、サーバの `sp_mda` プロシージャは `ct_connect` の実行時に呼び出されます。SQL コマンドを使用せずに text データのみまたは image データのみを送信する `ct_send_data` ルーチンがサーバでサポートされていないことが `sp_mda` で指定されている場合は、*type* パラメータを `CS_SEND_DATA_NOCMD` に設定した `ct_command` ルーチンの呼び出しは失敗します。

サーバが SQL コマンドを使用せずに text データまたは image データを受信できる場合、`sp_mda` は次の結果を返します。

パラメータ	値
mdinfo	"SENDDATA_NOCMD"
querytype	2
query	senddata no cmd

**注意** Adaptive Server では、SQL コマンドを使用せずに image データまたは text データを受信することはできません。

## クライアントへのデータの返送

アプリケーションは、ロー結果を返すことができるすべてのイベント・ハンドラから `text` または `image` データを返すことができます。アプリケーションは、ローのデータに含まれているカラムの数によって、いくつかの異なった手順で、出力される `text` または `image` データを処理します。カラムが 1 つしかなく、それが `text` または `image` カラムである場合には、アプリケーションは次の手順をとります。

- 1 `srv_descfmt` を使って、クライアントが `text` または `image` カラムを受け取るフォーマットを記述します。
- 2 `cmd` を `CS_SET` に設定して `srv_text_info` を呼び出して、`text` 全体の長さを設定します。
- 3 データをまとめてクライアントに送信するために、`srv_send_text` を呼び出します。

`text` カラムと `image` カラムの他にカラムがある場合、アプリケーションは次の手順に従わなければなりません。

- 1 `srv_descfmt` を使って、クライアントが `text` または `image` カラムを受け取るフォーマットを記述します。これは各カラムにつき一度ずつ呼び出されます。
- 2 `srv_bind` を使って、`Open Server` アプリケーションが情報を保存するローカル・プログラム変数のフォーマットと位置を記述します。これは各カラムにつき一度ずつ呼び出さなければなりません。
- 3 `srv_text_info` を呼び出すことで、テキスト・ポインタとタイムスタンプ情報を提供します。`cmd` を `CS_SET` に設定して、各 `text` または `image` カラムにつき一度ずつ呼び出さなければなりません。
- 4 `srv_xferdata` を使用してデータを転送します。このルーチンは、ローの数と同じ回数だけ呼び出さなければなりません。

`text` および `image` カラムの部分更新の詳細については、「[パラメータとロー・データの処理](#)」(152 ページ)を参照してください。

## 例

サンプル・プログラム `ctos.c` には、`text` データと `image` データを処理するコードが記述されています。

## データ型

Open Server は、広い範囲にわたるデータ型をサポートします。これらのデータ型は、CS-Library、Client-Library と共有されます。ほとんどの場合には、これらは Adaptive Server Enterprise データ型と直接対応します。

表 2-37 に、Open Server の型定義と、対応する型、対応する Adaptive Server Enterprise のデータ型を示します。各データ型に関する詳細情報は、表の後にあります。

このバージョンにおいては、下位互換性を保つために、2.0 Open Server データ型が含まれています。このバージョンでは、2.0 Server-Library ルーチンは 2.0 データ型を使用しなければなりません。表 2-37 は、今後の Open Server バージョンですべてのルーチンが使用しなければならない Open Server データ型をまとめたものです。

表 2-37 : データ型の概要

データ型	Open Client と Open Server の型定数	説明	対応する Open Client/Server の型定義	対応する Adaptive Server Enterprise のデータ型
binary 型	CS_BINARY_TYPE	バイナリ型	CS_BINARY	binary、varbinary
	CS_LONGBINARY_TYPE	長いバイナリ型	CS_LONGBINARY	なし
	CS_VARBINARY_TYPE	可変長バイナリ型	CS_VARBINARY	なし
bit 型	CS_BIT_TYPE	ビット型	CS_BIT	boolean
文字型	CS_CHAR_TYPE	文字型	CS_CHAR	char、varchar
	CS_LONGCHAR_TYPE	長い文字型	CS_LONGCHAR	NONE
	CS_VARCHAR_TYPE	可変長文字型	CS_VARCHAR	NONE
	CS_UNICHAR_TYPE	可変長または固定長文字型	CS_UNICHAR	unichar、univarchar
XML 型	CS_XML_TYPE	可変長文字型	CS_XML	xml

データ型	Open Client と Open Server の型定数	説明	対応する Open Client/Server の型定義	対応する Adaptive Server Enterprise のデータ型
datetime 型	CS_DATE_TYPE	4 バイトの日付データ型	CS_DATE	date
	CS_TIME_TYPE	4 バイトの時刻データ型	CS_TIME	time
	CS_DATETIME_TYPE	8 バイトの日時型	CS_DATETIME	datetime
	CS_DATETIME4_TYPE	4 バイトの日時型	CS_DATETIME4	smalldatetime
	CS_BIGDATETIME_TYPE	8 バイトのバイナリ型	CS_BIGDATETIME	bigdatetime
	CS_BIGTIME_TYPE	8 バイトのバイナリ型	CS_BIGTIME	bigtime
numeric 型	CS_TINYINT_TYPE	1 バイトの整数型	CS_TINYINT	tinyint
	CS_SMALLINT_TYPE	2 バイトの整数型	CS_SMALLINT	smallint
	CS_INT_TYPE	4 バイトの整数型	CS_INT	int
	CS_BIGINT_TYPE	8 バイトの整数型	CS_BIGINT	bigint
	CS_USMALLINT_TYPE	2 バイトの符号なし整数型	CS_USMALLINT	usmallint
	CS_UINT_TYPE	4 バイトの符号なし整数型	CS_UINT	uint
	CS_UBIGINT_TYPE	8 バイトの符号なし整数型	CS_UBIGINT	ubigint
	CS_DECIMAL_TYPE	10 進数型	CS_DECIMAL	decimal
	CS_NUMERIC_TYPE	数値型	CS_NUMERIC	numeric
	CS_FLOAT_TYPE	8 バイトの浮動小数点型	CS_FLOAT	float
	CS_REAL_TYPE	4 バイトの浮動小数点型	CS_REAL	real
LOB (ラージ・オブジェクト) ロケータ型	CS_TEXTLOCATOR_TYPE	Textlocator 型	CS_TEXTLOCATOR	textlocator
	CS_IMAGELOCATOR_TYPE	Imagelocator 型	CS_IMAGELOCATOR	imagelocator
	CS_UNITEXTLOCATOR_TYPE	符号なし可変長文字型	CS_UNITEXTLOCATOR	unitextlocator

データ型	Open Client と Open Server の型定数	説明	対応する Open Client/Server の型定義	対応する Adaptive Server Enterprise のデータ型
money 型	CS_MONEY_TYPE	8 バイトの通貨型	CS_MONEY	money
	CS_MONEY4_TYPE	4 バイトの通貨型	CS_MONEY4	smallmoney
text 型および image 型	CS_TEXT_TYPE	テキスト型	CS_TEXT	text
	CS_UNITEXT_TYPE	符号なし可変長文字型	CS_UNITEXT	unitext
	CS_IMAGE_TYPE	イメージ型	CS_IMAGE	image

## データ型を操作するルーチン

CS-Library には、データ型の操作に便利なルーチンがあります。その中には、次のものがあります。

- `cs_calc` : `decimal`、`float`、`money`、`numeric`、`real` のデータ型に対して算術演算を行います。
- `cs_cmp` : `datetime`、`decimal`、`float`、`money`、`numeric`、`real` のデータ型の比較を行います。
- `cs_convert` : ある型のデータ値を別のデータ型に変換します。
- `cs_dt_crack` : マシンが読み取れる日時値をユーザがアクセス可能なフォーマットに変換します。
- `cs_dt_info` : 各国言語の日時情報を取得します。

これらのルーチンについては、『Open Client/Server Common Libraries リファレンス・マニュアル』に記載されています。

## Open Server のデータ型

### binary 型

Open Server には、`CS_BINARY`、`CS_LONGBINARY`、`CS_VARBINARY` という 3 つの binary 型があります。

- `CS_BINARY` は、Adaptive Server Enterprise の binary データ型と `varbinary` データ型に対応します。つまり、Server-Library はサーバの binary 型と `varbinary` 型を `CS_BINARY` として解釈します。たとえば、`srv_descfmt` はクライアントからバイナリ・パラメータの記述を取得するときに、`CS_BINARY_TYPE` を返します。

`CS_BINARY` は、次のように定義されます。

```
typedef unsigned char    CS_BINARY;
```

- `CS_LOGBINARY` はどの Adaptive Server Enterprise データ型にも対応していませんが、一部の Open Server アプリケーションは `CS_LOGBINARY` をサポートします。アプリケーションでは、`CS_DATA_LBIN` 機能を使用して、Client-Library 接続で `CS_LOGBINARY` がサポートされているかどうかを確認できません。

`CS_LOGBINARY` 値の最大長は、2,147,483,647 バイトです。

`CS_LOGBINARY` の定義は次のとおりです。

```
typedef unsigned char    CS_LOGBINARY;
```

- `CS_VARBINARY` は、どの Adaptive Server Enterprise データ型にも対応しません。そのため、Open Server ルーチンは、`CS_VARBINARY_TYPE` を返しません。データ型が `CS_VARBINARY_TYPE` と記述された場合には、Open Server はそれを自動的に `NULL` が許される `CS_BINARY_TYPE` に変換してからクライアントに送ります。`CS_VARBINARY_TYPE` は、プログラム変数をバインドするときのみ使用可能です。`CS_VARBINARY` によって、プログラマは C 以外のプログラミング言語で Open Server のプログラムを書くことができます。一般的なサーバ・アプリケーションは、`CS_VARBINARY` を使用しません。

`CS_VARBINARY` の定義は次のとおりです。

```
typedef struct _cs_varybin
{
    CS_SMALLINT    len;
    CS_BYTE        array[CS_MAX_CHAR];
} CS_VARBINARY;
```

各パラメータの意味は次のとおりです。

- `len` はバイナリ配列の長さです。
- `array` は配列そのものです。



## bit 型

Open Server でサポートされているビット型は、CS\_BIT のみです。このデータ型には、0 または 1 のサーバ・ビット値 (またはブール値) が格納されます。他の型を bit 型に変換すると、ゼロ以外の値はすべて 1 に変換されます。

```
typedef unsigned char    CS_BIT;
```

## 文字型

Open Server には、CS\_CHAR、CS\_LONGCHAR、CS\_VARCHAR、CS\_UNICHAR の 4 つの character 型があります。

- CS\_CHAR は、Adaptive Server Enterprise の char データ型と varchar データ型に対応します。つまり、Server-Library はサーバの char データ型と varchar データ型を CS\_CHAR として解釈します。たとえば、srv\_descfmt はクライアントから文字パラメータの記述を取得するときに、CS\_CHAR\_TYPE を返します。

CS\_CHAR の定義は次のとおりです。

```
typedef char    CS_CHAR;
```

- CS\_LONGCHAR はどの Adaptive Server Enterprise データ型にも対応していませんが、Client-Library アプリケーションによっては CS\_LONGCHAR をサポートするものもあります。アプリケーションでは、CS\_DATA\_LCHAR 機能を使用して、Client-Library 接続が CS\_LONGCHAR をサポートしているかどうかを調べることができます。

CS\_LONGCHAR 値は、最大 2,147,483,647 バイトの長さをサポートします。CS\_LONGCHAR の定義は次のとおりです。

```
typedef unsigned char    CS_LONGCHAR;
```

- CS\_VARCHAR は Adaptive Server Enterprise のどのデータ型とも対応しません。このため、Open Server ルーチンは、CS\_VARCHAR\_TYPE を返しません。データ型が CS\_VARCHAR\_TYPE と記述された場合には、Open Server はそれを自動的に null 入力可能な CS\_CHAR\_TYPE に変換してからクライアントに送ります。CS\_VARCHAR\_TYPE は、プログラム変数をバインドするときのみ使用できます。CS\_VARCHAR によって、プログラムは C 以外のプログラミング言語で Open Server のプログラムを書くことができます。一般的なサーバ・アプリケーションは、CS\_VARCHAR を使用しません。

CS\_VARCHAR の定義は次のとおりです。

```
typedef struct _cs_varchar
{
    CS_SMALLINT    len;
    CS_BYTE        str[CS_MAX_CHAR];
} CS_VARCHAR;
```

各パラメータの意味は次のとおりです。

- *len* は文字列の長さです。
- *str* は文字列です。*str* は `null` で終了する文字列ではないことに注意してください。
- `CS_UNICHAR` は、Adaptive Server Enterprise の `unichar` 固定幅および `univarchar` 可変幅のデータ型に対応します。`CS_UNICHAR` は、`CS_CHAR` データ型が使用されるどの場所でも使用できる共有 C プログラミング・データ型です。`CS_UNICHAR` データ型は、2 バイトの Unicode UTF-16 フォーマットで文字データを保存します。`CS_UNICHAR` の定義は次のとおりです。

```
typedef unsigned char    CS_UNICHAR;
```

## XML 型

`CS_XML` は、Adaptive Server Enterprise の `xml` 可変長データ型に直接対応します。`CS_XML` は、XML ドキュメントとそのコンテンツを表し、`CS_TEXT` と `CS_IMAGE` を使用できるところであればどこでも使用できます。

`CS_XML` の定義は次のとおりです。

```
typedef unsigned char    CS_XML
```

## datetime 型

Open Server は 6 つの `datetime` 型、`CS_DATE`、`CS_TIME`、`CS_DATETIME`、`CS_DATETIME4`、`CS_BIGDATETIME`、`CS_BIGTIME` をサポートします。これらのデータ型は、8 バイトまたは 4 バイトの `datetime` 値を保持します。

CS\_BIGDATETIME および CS\_BIGTIME データ型は、マイクロ秒の精度の *time* データを提供します。これらのデータ型には 8 バイトのバイナリ値が格納されます。これらのデータ型はそれぞれ、CS\_DATETIME データ型および CS\_TIME データ型に似ています。CS\_BIGDATETIME データ型は、CS\_DATETIME データ型を使用する場所ならどこでも使用可能です。CS\_BIGTIME データ型は、CS\_TIME データ型を使用する場所ならどこでも使用可能です。CS\_DATETIME データ型および CS\_TIME データ型に適用できるすべての Open Client および Open Server ルーチンは、CS\_BIGDATETIME データ型および CS\_BIGTIME データ型にも適用できます。

Open Server アプリケーションは、CS-Library ルーチンの *cs\_dt\_crack* を使って日時構造体から日付要素 (年、月、日など) を抽出できます。

- CS\_DATETIME は、Adaptive Server Enterprise の *datetime* データ型に対応しています。CS\_DATETIME の有効値は 1753 年 1 月 1 日から 9999 年 12 月 31 日の範囲で、精度は 1 秒の 300 分の 1 (3.33 ミリ秒) です。

```
typedef struct _cs_datetime
{
    CS_INT          dtdays;
    CS_INT          dttime;
} CS_DATETIME;
```

各パラメータの意味は次のとおりです。

- *dtdays* は 1900 年 1 月 1 日から数えた日数です。
- *dttime* は、深夜 0 時からの 300 分の 1 秒の数です。
- CS\_DATETIME4 は、Adaptive Server Enterprise の *smalldatetime* データ型に対応しています。CS\_DATETIME4 の有効値は、1900 年 1 月 1 日から 2079 年 6 月 6 日の範囲で、精度は 1 分です。

```
typedef struct _cs_datetime4
{
    unsigned short  days;
    unsigned short  minutes;
} CS_DATETIME4;
```

各パラメータの意味は次のとおりです。

- *days* は 1900 年 1 月 1 日から数えた日数です。
- *minutes* は、深夜 0 時からの分数です。

- `CS_DATE` は、Adaptive Server Enterprise の `date` データ型に対応します。有効な `CS_DATE` 値の範囲は、1753 年 1 月 1 日から 9999 年 12 月 31 日までです。

```
typedef struct _cs_date
{
    CS_INT          days;
} CS_DATE;
```

`days` は、1900 年 1 月 1 日からの日数です。

- `CS_TIME` は、Adaptive Server Enterprise の `time` データ型に対応します。有効な `CS_TIME` 値の範囲は、12:00:00.000 から 11:59:59.999 までで、精度は 300 分の 1 秒 (3.33 ミリ秒) です。

```
typedef struct _cs_time
{
    CS_INT          time;
} CS_TIME;
```

`time` は、深夜 0 時からの 300 分の 1 秒の数です。

- `CS_BIGDATETIME` は、Adaptive Server Enterprise のデータ型 `bigdatetime` に対応し、0000 年 1 月 1 日の 00:00:00.000000 から経過したマイクロ秒数を格納します。有効な `CS_BIGDATETIME` 値の範囲は、0001 年 1 月 1 日の 00:00:00.000000 から 9999 年 12 月 31 日の 23:59:59.999999 までです。

---

**注意** 0000 年 1 月 1 日の 00:00:00.000000 は、マイクロ秒数のカウントが開始される基本の値です。0001 年 1 月 1 日の 00:00:00.000000 より前の値は無効です。

---

`CS_BIGDATETIME` の定義は、`cstypes.h` にあります。

```
typedef CS_UBIGINT CS_BIGDATETIME;
```

- `CS_BIGTIME` は、Adaptive Server Enterprise のデータ型 `bigtime` に対応し、当日の午前 0 時ちょうどから経過したマイクロ秒数を示します。有効な `CS_BIGTIME` 値の範囲は、00:00:00.000000 から 23:59:59.999999 までです。`CS_BIGTIME` の定義は、`cstypes.h` にあります。

```
typedef CS_UBIGINT CS_BIGTIME;
```

- `CS_BIGDATETIME` データ型および `CS_BIGTIME` データ型は、基本となるクライアント・プラットフォームのネイティブのバイト順序 (エンディアン) のクライアントに示されます。必要であればサーバで行われるバイト・スワッピングは、クライアントにデータが送られる前、またはクライアントからのデータを受け取った後に行われます。

## datetime の最小値と最大値

次の表に、datetime 型の最小値と最大値を示します。

表 1 : datetime の最小値と最大値

データ型	最小値	最大値
<code>CS_BIGDATETIME</code>	January 1, 0001 00:00:00.000000	December 31, 9999 23:59:59.999999
<code>CS_BIGTIME</code>	00:00:00.000000	23:59:59.999999
<code>CS_DATE</code>	January 1, 0001	December 31, 9999
<code>CS_DATETIME</code>	January 1, 1753 00:00:00.000	December 31, 9999 23:59:59.999
<code>CS_DATETIME4</code>	January 1, 1900 00:00:00.000	June 6, 2079 23:59:59.999
<code>CS_TIME</code>	00:00:00.000	23:59:59.999

## 整数値型

Open Server は次の 7 つの integer 型、`CS_TINYINT`、`CS_SMALLINT`、`CS_INT`、`CS_BIGINT`、`CS_USMALLINT`、`CS_UINT`、`CS_UBIGINT` をサポートしています。

大半のプラットフォームで、`CS_TINYINT` は 1 バイトの整数、`CS_SMALLINT` は 2 バイトの整数、`CS_INT` は 4 バイトの整数、`CS_BIGINT` は 8 バイトの整数、`CS_USMALLINT` は符号なし 2 バイトの整数、`CS_UINT` は符号なし 4 バイトの整数、`CS_UBIGINT` は符号なし 8 バイトの整数です。

```
typedef unsigned char    CS_TINYINT;
typedef short           CS_SMALLINT;
typedef int             CS_INT;
typedef long long       CS_BIGINT;
typedef unsigned char   CS_USMALLINT;
typedef unsigned int    CS_UINT;
typedef unsigned long long CS_UBIGINT;
```

**real、float、numeric、decimal 型**

- CS\_REAL は、Adaptive Server Enterprise のデータ型 *real* に対応しています。これは、プラットフォームに依存する C 言語の *float* 型として実装されています。

```
typedef float                CS_REAL;
```

---

**注意** 6桁精度の *bigint* データ型または *ubigint* データ型を *real* のデータ型に変換する場合、次の最大値および最小値に注意してください。

- $-9223370000000000000.0 < bigint < 9223370000000000000.0$
- $0 < ubigint < 18446700000000000000.0$

これらの範囲外の値により、オーバーフロー・エラーが発生します。

- CS\_FLOAT は、Adaptive Server Enterprise の *float* データ型に対応しています。これは、プラットフォームに依存する C 言語の *double* 型として実装されています。

```
typedef double              CS_FLOAT;
```

---

**注意** 15桁精度の *bigint* データ型または *ubigint* データ型を *float* のデータ型に変換する場合、次の最大値および最小値に注意してください。

- $-9223372036854770000.0 < bigint < 9223372036854770000.0$
- $0 < ubigint < 18446744073709500000.0$

これらの範囲外の値により、オーバーフロー・エラーが発生します。

- CS\_NUMERIC と CS\_DECIMAL は、Adaptive Server Enterprise のデータ型 *numeric* と *decimal* に対応します。これらは、精度と位取りを持った数値に対して、プラットフォームに依存しないサポートを提供します。

Adaptive Server Enterprise の *numeric* データ型と *decimal* データ型は等価で、CS\_DECIMAL は CS\_NUMERIC として定義されます。

```
typedef struct  _cs_numeric
{
    CS_BYTE      precision;
    CS_BYTE      scale;
```

```

        CS_BYTE                                array[CS_MAX_NUMLEN];
    } CS_NUMERIC;

typedef CS_NUMERIC                            CS_DECIMAL;

```

各パラメータの説明は、次のとおりです。

- *precision* は、数値の精度です。 *precision* の有効値は、 `CS_MIN_PREC` から `CS_MAX_PREC` の範囲です。デフォルトの精度は `CS_DEF_PREC` です。 `CS_MIN_PREC`、 `CS_MAX_PREC`、 `CS_DEF_PREC` はそれぞれ、精度の最小値、最大値、デフォルト値を示します。
- *scale* は数値の位取りです。 *scale* の有効値は、 `CS_MIN_SCALE` から `CS_MAX_SCALE` の範囲です。デフォルトの位取りは `CS_DEF_SCALE` です。 `CS_MIN_SCALE`、 `CS_MAX_SCALE`、 `CS_DEF_SCALE` はそれぞれ、位取りの最小値、最大値、デフォルト値を示します。
- *scale* は、 *precision* 以下でなければなりません。

`CS_DECIMAL` 型の精度 (*precision*) と位取り (*scale*) のデフォルト値は、 `CS_NUMERIC` 型と同じです。

## ラージ・オブジェクト・ロケータのデータ型

Open Server は LOB (ラージ・オブジェクト) ロケータをサポートしています。LOB ロケータには、データ自体ではなく、Adaptive Server 内の LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。

`CS_LOCATOR` データ型は LOB ロケータをサポートしています。`CS_LOCATOR` 型は、ロケータの値とオプションのプリフェッチされたデータを格納する `opaque` データ型です。受信ロケータを `CS_LOCATOR` 変数にバインドする前に、`cs_locator_alloc()` を使用してこの変数のメモリを割り付けてください。そうしないとエラーが発生します。変数が無用になった場合は、`cs_locator_drop()` を使用してそのメモリを解放します。

`CS_LOCATOR` 変数は再使用できます。ただし、Adaptive Server 内の現在のロケータ値は、トランザクションが終了するまでのみ有効です。`CS_LOCATOR` データ型には、`CS_TEXTLOCATOR_TYPE`、`CS_IMAGELOCATOR_TYPE`、および `CS_UNITEXTLOCATOR_TYPE` という 3 つの型定数があります。

## money 型

Open Server では、CS\_MONEY と CS\_MONEY4 の 2 つの money 型がサポートされます。これらのデータ型は、それぞれ 8 バイトと 4 バイトの money 型を保持できます。

- CS\_MONEY は、Adaptive Server Enterprise の money データ型に対応しています。有効な CS\_MONEY 値の範囲は、次のとおりです。  
+/- \$922,337,203,685,477.5807:

```
typedef struct _cs_money
{
    CS_INT          mnyhigh;
    CS_UINT         mnylow;
} CS_MONEY;
```

- CS\_MONEY4 は、Adaptive Server Enterprise の smallmoney データ型に対応しています。CS\_MONEY4 の有効値は、-\$214,748.3648 から +\$214,748.3647 の範囲です。

```
typedef struct _cs_money4
{
    CS_INT    mny4;
} CS_MONEY4;
```

## security 型

Open Server では、型定数 CS\_BOUNDARY\_TYPE と CS\_SENSITIVITY\_TYPE を定義することで、Secure Adaptive Server Enterprise の boundary と sensitivity データ型をサポートします。

これらの型定数は、他の Open Server の型定数とは異なり、似たような名前の型定義に対応しません。これらは CS\_CHAR に対応します。

つまり、Open Server ルーチンがカラムや変数のデータ型を記述するために CS\_BOUNDARY\_TYPE と CS\_SENSITIVITY\_TYPE を受け入れて返す場合も、対応するプログラム変数の型は CS\_CHAR でなければなりません。

たとえば、アプリケーションが、CS\_DATAFMT 構造体の *datatype* フィールドを CS\_SENSITIVITY\_TYPE に設定して *srv\_bind* を呼び出す場合、データがバインドされるプログラム変数の型は CS\_CHAR でなければなりません。



## text 型および image 型

Open Server は、text データ型 CS\_TEXT と CS\_UNITEXT、image データ型 CS\_IMAGE をサポートします。

- CS\_TEXT は、最大 2,147,483,647 バイトの印刷可能文字データを格納する可変長カラムを定義する、サーバのデータ型 text に対応しています。CS\_TEXT は符号なしの文字型として定義されます。

```
typedef unsigned char      CS_TEXT;
```
- CS\_UNITEXT は、Adaptive Server Enterprise の unitext 可変長データ型に直接対応します。CS\_UNITEXT と CS\_TEXT は、共通の構文と語義を使用します。ただし、CS\_UNITEXT では、文字データが 2 バイト UTF-16 形式でコード化されます。CS\_UNITEXT は、CS\_TEXT が使用されるどの場所でも使用できます。CS\_UNITEXT 文字列パラメータの最大長は、CS\_TEXT の最大長の半分です。

CS\_UNITEXT の定義は次のとおりです。

```
typedef unsigned short      CS_UNITEXT;
```

- CS\_IMAGE は、最大 2,147,483,647 バイトのバイナリ・データを格納する可変長カラムを定義する、サーバのデータ型 image に対応しています。CS\_IMAGE は符号なしの文字型として定義されます。

```
typedef unsigned char      CS_IMAGE;
```



## ルーチン

この章は、各 Server-Library ルーチンについて説明します。

ルーチン	説明	ページ
<a href="#">srv_alloc</a>	メモリを割り付けます。	248
<a href="#">srv_alt_bind</a>	計算ローのカラムにソース・データを記述し、バインドします。	250
<a href="#">srv_alt_descfmt</a>	計算ロー・カラムの集合演算子と、クライアントに返されるカラム・データのフォーマットを記述します。	255
<a href="#">srv_alt_header</a>	計算ローのロー識別子および <code>bylist</code> を記述します。	259
<a href="#">srv_alt_xferdata</a>	クライアントに計算ローを送ります。	261
<a href="#">srv_bind</a>	カラムまたはパラメータにプログラム変数を記述し、バインドします。	263
<a href="#">srv_bmove</a>	バイトを、1つのメモリ・ロケーションから別のメモリ・ロケーションにコピーします。	269
<a href="#">srv_bzero</a>	メモリ・ロケーションの内容を 0 に設定します。	271
<a href="#">srv_callback</a>	スレッドに対して状態遷移ハンドラをインストールします。	272
<a href="#">srv_capability</a>	Open Server がプラットフォーム依存のサービスをサポートするかどうかを決定します。	276
<a href="#">srv_capability_info</a>	クライアント接続に関する機能情報を定義または取得します。	278
<a href="#">srv_createmsgq</a>	メッセージ・キューを作成します。	282
<a href="#">srv_createmutex</a>	相互排他セマフォを作成します。	284
<a href="#">srv_createproc</a>	非クライアントのイベント駆動型スレッドを作成します。	286
<a href="#">srv_cursor_props</a>	(15.0 より前のアプリケーションの場合) 現在のカーソルに関する情報を取得または設定します。	288
<a href="#">srv_cursor_props2</a>	(15.0 以降のアプリケーションの場合) 現在のカーソルに関する情報を取得または設定します。	291

ルーチン	説明	ページ
<a href="#">srv_dbg_stack</a>	スレッドのコール・スタックを表示します。	<a href="#">292</a>
<a href="#">srv_dbg_switch</a>	デバッグのために、もう1つのスレッド・コンテキストを一時的にリストアします。	<a href="#">295</a>
<a href="#">srv_define_event</a>	ユーザ・イベントを定義します。	<a href="#">295</a>
<a href="#">srv_deletemsgq</a>	メッセージ・キューを削除します。	<a href="#">298</a>
<a href="#">srv_deletemutex</a>	<a href="#">srv_createmutex</a> を使用して作成したミューテックスを削除します。	<a href="#">300</a>
<a href="#">srv_descfmt</a>	クライアントとやり取りするカラムまたはパラメータの記述を、記述または取得します。	<a href="#">301</a>
<a href="#">srv_dynamic</a>	クライアントの動的 SQL コマンドを読み込む、またはコマンドに応答します。	<a href="#">306</a>
<a href="#">srv_envchange</a>	環境の変化をクライアントに通知します。	<a href="#">310</a>
<a href="#">srv_event</a>	スレッドの要求処理キューに、イベント要求を追加します。	<a href="#">313</a>
<a href="#">srv_event_deferred</a>	非同期イベントの結果として、スレッドのイベント・キューにイベント要求を追加します。	<a href="#">315</a>
<a href="#">srv_free</a>	割り付けられているメモリを解放します。	<a href="#">318</a>
<a href="#">srv_freeserveraddrs</a>	<a href="#">srv_getserverbyname</a> によって割り付けられたメモリを解放します。	<a href="#">319</a>
<a href="#">srv_get_data</a>	<a href="#">text</a> 、 <a href="#">unitext</a> 、または <a href="#">image</a> パラメータ・ストリームをまとまった単位でクライアントから読み込みます。	<a href="#">320</a>
<a href="#">srv_get_text</a>	連続したデータとして、 <a href="#">text</a> または <a href="#">image</a> のデータ・ストリームをクライアントから読み込みます。	<a href="#">321</a>
<a href="#">srv_getloginfo</a>	リモート・サーバとのバスルー接続の準備のために、クライアント・スレッドからログイン情報を取得します。	<a href="#">324</a>
<a href="#">srv_getmsgq</a>	メッセージ・キューから次のメッセージを取得します。	<a href="#">326</a>
<a href="#">srv_getobjid</a>	指定の名前のメッセージ・キューまたはミューテックスのオブジェクト ID を検索します。	<a href="#">329</a>

ルーチン	説明	ページ
<a href="#">srv_getobjname</a>	指定した名前を持つ、メッセージ・キューまたはミューテックスの名前を取得します。	<a href="#">331</a>
<a href="#">srv_getserverbyname</a>	<i>server_name</i> の接続情報を返し、必要に応じてメモリを割り付けます。	<a href="#">334</a>
<a href="#">srv_handle</a>	Open Server アプリケーションにイベント・ハンドラをインストールします。	<a href="#">335</a>
<a href="#">srv_init</a>	Open Server アプリケーションを初期化します。	<a href="#">338</a>
<a href="#">srv_langcpy</a>	クライアントの言語要求を、アプリケーション・バッファにコピーします。	<a href="#">340</a>
<a href="#">srv_langlen</a>	言語要求バッファの長さを返します。	<a href="#">343</a>
<a href="#">srv_lockmutex</a>	ミューテックスをロックします。	<a href="#">344</a>
<a href="#">srv_log</a>	Open Server アプリケーション・ログ・ファイルへメッセージを書き込みます。	<a href="#">347</a>
<a href="#">srv_mask</a>	SRV_MASK_ARRAY 構造体のビットを初期化、チェック、設定、またはクリアします。	<a href="#">349</a>
<a href="#">srv_msg</a>	メッセージ・データ・ストリームを送信または受信します。	<a href="#">351</a>
<a href="#">srv_negotiate</a>	ネゴシエーション・ログイン情報をクライアントに送信およびクライアントから受信します。	<a href="#">355</a>
<a href="#">srv_numparams</a>	現在のクライアント・コマンドに含まれているパラメータの数を返します。	<a href="#">363</a>
<a href="#">srv_options</a>	オプション情報をクライアントに送信、またはクライアントから受信します。	<a href="#">365</a>
<a href="#">srv_orderby</a>	クライアントに <i>order-by</i> リストを返します。	<a href="#">372</a>
<a href="#">srv_poll (UNIX のみ)</a>	一連のオープン・ストリームのファイル記述子の I/O イベントをチェックします。	<a href="#">374</a>
<a href="#">srv_props</a>	Open Server プロパティを定義または取得します。	<a href="#">377</a>
<a href="#">srv_putmsgq</a>	メッセージ・キューにメッセージを入れます。	<a href="#">385</a>
<a href="#">srv_realloc</a>	メモリの再割り付けを行います。	<a href="#">387</a>

ルーチン	説明	ページ
<a href="#">srv_recvpass thru</a>	クライアントからプロトコル・パケットを受信します。	389
<a href="#">srv_regcreate</a>	レジスタード・プロシージャの登録を完了します。	391
<a href="#">srv_regdefine</a>	プロシージャを登録するプロセスを開始します。	393
<a href="#">srv_regdrop</a>	プロシージャの登録を解除します。	396
<a href="#">srv_regexec</a>	レジスタード・プロシージャを実行します。	398
<a href="#">srv_reginit</a>	レジスタード・プロシージャの実行を開始します。	400
<a href="#">srv_reglist</a>	Open Server に登録されているすべてのプロシージャのリストを取得します。	403
<a href="#">srv_reglistfree</a>	以前に割り付けられた <b>SRV_PROCLIST</b> 構造体を解放します。	405
<a href="#">srv_regnowatch</a>	レジスタード・プロシージャの通知リストから、クライアント・スレッドを削除します。	406
<a href="#">srv_regparam</a>	定義されているレジスタード・プロシージャに対してパラメータを記述する、またはレジスタード・プロシージャの実行に対してデータを提供します。	408
<a href="#">srv_regwatch</a>	指定されたプロシージャの通知リストに、クライアント・スレッドを追加します。	411
<a href="#">srv_regwatchlist</a>	クライアント・スレッドが通知要求待ちになっている、すべてのレジスタード・プロシージャのリストを返します。	415
<a href="#">srv_rpcdb</a>	現在のリモート・プロシージャ・コールで指定されているデータベース要素を返します。	416
<a href="#">srv_rpcname</a>	現在のリモート・プロシージャ・コールの名前から、RPC 名に当たる部分を返します。	418
<a href="#">srv_rpcnumber</a>	現在のリモート・プロシージャで指定されている番号要素を返します。	421

ルーチン	説明	ページ
<a href="#">srv_rpcoptions</a>	現在のリモート・プロシージャ・コールのランタイム・オプションを返します。	<a href="#">423</a>
<a href="#">srv_rpcowner</a>	現在のリモート・プロシージャ・コールで指定されている所有者要素を返します。	<a href="#">424</a>
<a href="#">srv_run</a>	Open Server アプリケーションを開始します。	<a href="#">426</a>
<a href="#">srv_s_ssl_local_id</a>	ローカル ID (認証) ファイルへのパスを指定するために使用します。	<a href="#">427</a>
<a href="#">srv_select (UNIX のみ)</a>	指定の I/O オペレーションに対してファイル記述子の準備ができていかどうかをチェックします。	<a href="#">428</a>
<a href="#">srv_send_ctlinfo</a>	Client-Library に制御メッセージを送信します。	<a href="#">431</a>
<a href="#">srv_send_data</a>	複数のカラムがあるローをクライアントに転送します。	<a href="#">433</a>
<a href="#">srv_send_text</a>	連続したデータとして、text または image のデータ・ストリームをクライアントに送ります。	<a href="#">437</a>
<a href="#">srv_senddone</a>	クライアントに結果完了メッセージを送るか、結果の一部をフラッシュします。	<a href="#">440</a>
<a href="#">srv_sendinfo</a>	クライアントにエラーメッセージ、または情報メッセージを送ります。	<a href="#">445</a>
<a href="#">srv_sendpassthru</a>	クライアントにプロトコル・パケットを送ります。	<a href="#">449</a>
<a href="#">srv_sendstatus</a>	クライアントにステータス値を送信します。	<a href="#">452</a>
<a href="#">srv_setcolotype</a>	カラムと関連付けられるユーザ・データ型を指定します。	<a href="#">453</a>
<a href="#">srv_setcontrol</a>	カラムのユーザ制御またはフォーマット情報を記述します。	<a href="#">455</a>
<a href="#">srv_setloginfo</a>	リモート・サーバからクライアントにプロトコルのフォーマット情報を返します。	<a href="#">458</a>
<a href="#">srv_setpri</a>	スレッドのスケジューリング優先順位を変更します。	<a href="#">460</a>

ルーチン	説明	ページ
<a href="#">srv_signal (UNIX のみ)</a>	SIGIO または SIGURG シグナル用に <code>signal</code> と同じインタフェースを使用した UNIX シグナル・ハンドラをインストールします。	462
<a href="#">srv_sleep</a>	現在実行中のスレッドを休止します。	464
<a href="#">srv_spawn</a>	サービス・スレッドを割り付けます。	468
<a href="#">srv_symbol</a>	Open Server のトークン値を読み込み可能な文字列に変換します。	471
<a href="#">srv_tabcolname</a>	ブラウズ・モードの結果カラムに結果テーブルを関連付けます。	476
<a href="#">srv_tabname</a>	一連のブラウズ・モードの結果と関連付けられたテーブルの名前を提供します。	478
<a href="#">srv_termproc</a>	スレッドの実行を中止します。	480
<a href="#">srv_text_info</a>	<code>text</code> または <code>image</code> データの記述を設定または取得します。	482
<a href="#">srv_thread_props</a>	スレッド・プロパティを定義し、取得します。	485
<a href="#">srv_timedsleep</a>	イベントが通知されるまでスリープします。	490
<a href="#">srv_unlockmutex</a>	ミューテックスのロックを解除します。	494
<a href="#">srv_version</a>	アプリケーションが使用している <code>Server-Library</code> のバージョンを定義し、アプリケーションのデフォルトの使用言語と文字セットを定義します。	495
<a href="#">srv_wakeup</a>	スリープ中のスレッドの実行を可能にします。	497
<a href="#">srv_xferdata</a>	クライアントにパラメータやデータを送信するか、またはクライアントからパラメータやデータを受信します。	500
<a href="#">srv_yield</a>	他のスレッドの実行を可能にします。	503

## srv\_alloc

説明

メモリを割り付けます。

構文

```
CS_VOID *srv_alloc(size)
CS_INT size;
```



パラメータ *size*  
割り付けるバイト数です。

戻り値 **表 3-1 : 戻り値 (srv\_alloc)**

戻り値	意味
新しく割り付けられた領域へのポインタ	新しい領域のロケーション。
null の CS_VOID ポインタ	Open Server が <i>size</i> バイトを割り付けできなかった。

例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
CS_RETCODE      ex_srv_alloc PROTOTYPE((
CS_BYTE        **bpp,
CS_INT         size
));
/*
** EX_SRV_ALLOC
**
** Example routine to allocate the specified amount of memory
** using srv_alloc.
**
** Arguments:
**   bpp      Return pointer to allocated memory here.
**   size     Amount of memory to allocate.
**
** Returns:
**
**   CS_SUCCEEDED      Memory was allocated successfully.
**   CS_FAIL           An error was detected.
**/
CS_RETCODE      ex_srv_alloc(bpp, size)
CS_BYTE        **bpp;
CS_INT         size;
{
    /* Initialization.*/
    *bpp = (CS_BYTE *)NULL;

    /*
    ** Allocate size number of bytes.
    **/
    if((*bpp = (CS_BYTE *)srv_alloc(size)) == (CS_BYTE *)NULL)
    {
```

```

        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

/*
** Allocate size number of bytes.
*/
if ((*bpp = (CS_BYTE *)srv_alloc(size)) == (CS_BYTE *)NULL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

使用方法

- `srv_alloc` は、メモリを動的に割り付けます。このルーチンは、*size* バイト (それだけのバイト数が使用可能な場合) に、ポインタを返します。
- `srv_alloc` を使用して割り付けられたメモリは、`srv_free` を呼び出して解放します。
- 標準の C メモリ割り付けルーチンを使用する場合は、`srv_alloc` を使用します。
- 現行バージョンでは、`srv_alloc` は、C ルーチンの `malloc` を呼び出します。しかし、Open Server アプリケーションでは、`srv_props` ルーチンを使用して独自のメモリ管理ルーチンをインストールすることができます。ユーザによりインストールされたルーチンのパラメータの受け渡しに関する規則は、`malloc` の規則と同一でなければなりません。ユーザによりインストールされたルーチンを使用できるようにアプリケーションが設定されていない場合は、Open Server は `malloc` を呼び出します。

参照

[srv\\_free](#), [srv\\_props](#), [srv\\_realloc](#)

## srv\_alt\_bind

説明

計算ローのカラムにソース・データを記述し、バインドします。

構文

```
CS_RETCODE srv_alt_bind(spp, altid, item, osfmtp,
                        varaddr, varlenp, indp)
```

```
SRV_PROC    *spp;
CS_INT      altid;
```

```

CS_INT          item;
CS_DATAFMT     *osfntp;
CS_BYTE        *varaddrp;
CS_INT          *varlenp;
CS_SMALLINT    *indp;

```

## パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*altid*

この計算カラムが含まれている計算ロー用のユニークな識別子です。*altid* は、*srv\_alt\_header* を使って定義されます。

*item*

計算ロー内のカラムの番号です。カラム番号は、1 から始まります。

*osfntp*

*CS\_DATAFMT* 構造体を指すポインタです。この構造体は、アプリケーション・プログラム変数に含まれている計算ロー・カラム・データのフォーマットを記述します。

*varaddrp*

送信データがバインドされているプログラム変数へのポインタです。

*varlenp*

\**varaddrp* の長さが格納されているプログラム変数へのポインタです。

*indp*

null 値のインジケータが格納されているバッファへのポインタです。次の表に、\**indp* に指定できる値を示します。

表 3-2 : *indp* の値 (*srv\_alt\_bind*)

値	意味
<i>CS_NULLDATA</i>	カラム・データは null。
<i>CS_GOODDATA</i>	カラム・データは null ではない。

*indp* が NULL の場合、カラム・データは有効、つまり、NULL ではないとみなされます。

## 戻り値

表 3-3 : 戻り値 (*srv\_alt\_bind*)

戻り値	意味
<i>CS_SUCCEED</i>	ルーチンが正常に終了した。
<i>CS_FAIL</i>	ルーチンが失敗。

## 例

```

#include          <ospublic.h>
/*

```

```

** Local prototype
*/
CS_RETCODE          ex_srv_alt_bind PROTOTYPE((
SRV_PROC            *spp,
CS_INT              altid,
CS_VOID             *sump
));
/*
** EX_SRV_ALT_BIND
**
** Example routine to describe and bind the source data for
** a compute row column.This example binds a value which
** is the sum of the first column of row data.
**
** Arguments:
** spp - A pointer to an internal thread control structure.
**      The thread must be an active client thread that
**      can handle row data.
**
** altid - The id for this compute row.
**
** sump - A pointer to the variable which will contain
**        the sum of the first column of row data.
**
** Returns:
** CS_SUCCEED - Compute row column was successfully bound.
** CS_FAIL - An error was detected.
**
*/
CS_RETCODE          ex_srv_alt_bind(spp, altid, sump)
SRV_PROC            *spp;
CS_INT              altid;
CS_VOID             *sump;
{
    CS_DATAFMT compute_colfmt;
    /*
    **Format for this compute column.
    */
    CS_INT          namelen;
    /*
    **Length of compute column name
    */

    CS_INT          compute_colnum;
    /*
    ** The column number for this compute column.
    */

```

```
CS_SMALLINT      indicator;
/*
**      Null indicator.
**/
CS_INT           sumlen;
/*
**      Length of the compute value
**/
CS_RETCODE       result;
/*
**Return value from srv_alt_bind.
**/

/*
** Initialize the compute column's data format.This compute
** column represents a sum of the first column of data.
**/
namelen = 3;
srv_bmove( "sum" , compute_colfmt.name, namelen);
compute_colfmt.namelen = namelen;
compute_colfmt.datatype = CS_INT_TYPE;
compute_colfmt.format = CS_FMT_UNUSED;
compute_colfmt.maxlength = sizeof(CS_INT);
compute_colfmt.scale = 0;

compute_colfmt.precision = CS_DEF_PREC;
compute_colfmt.status = 0;
compute_colfmt.count = 0;
compute_colfmt.usertype = 0;
compute_colfmt.locale = (CS_LOCALE *)NULL;

/*
** Perform the bind
**/
compute_colnum = 1;
indicator = CS_GOODDATA;
sumlen = sizeof(CS_INT);

result = srv_alt_bind(spp, altid, compute_colnum,
                    &compute_colfmt, sump, &sumlen, &indicator);
return (result);
}
```

使用法

- 計算ローの情報を返す Adaptive Server Enterprise の機能を模擬的に  
行うアプリケーションのみが、`srv_alt_bind` を呼び出す必要があります。`srv_alt_bind` は、Adaptive Server Enterprise へのゲートウェイ  
としての役割を果たしているアプリケーションにとって便利です。
- `srv_alt_bind` は、計算ロー・カラムのデータが保存されているア  
プリケーション・プログラム変数のフォーマットを記述します。ア  
プリケーションは、計算ローの各カラムで一度ずつ `srv_alt_bind`  
を呼び出さなければなりません。
- `srv_alt_bind` ルーチンは、次の表に示す `CS_DATAFMT` フィールド  
に対して、読み取り (`CS_GET`) や設定 (`CS_SET`) を行います。他の  
すべてのフィールドは、`srv_alt_bind` には未定義です ( “`osfmtp`” は  
構造体を指すポインタであることに注意してください)。

表 3-4 : 使用される `CS_DATAFMT` フィールド (`srv_alt_bind`)

フィールド	CS_SET	CS_GET
<code>osfmtp</code> → <code>datatype</code>	アプリケーション・プロ グラム変数のデータ型	アプリケーション・プロ グラム変数のデータ型
<code>osfmtp</code> → <code>maxlength</code>	未使用	プログラム変数の最大長
<code>osfmtp</code> → <code>count</code>	0 または 1。	0 または 1。

- `osfmtp` が記述するフォーマットと `srv_alt_descfmt` (`clfmtp`) で設定さ  
れたクライアント・フォーマットが異なる場合には、Open Server  
は自動的にデータをクライアント・フォーマットに変換します。
- 計算結果セットは、1 つのローしか含みません。ただし、ア  
プリケーションは、それぞれが独自の `altid` を持った複数の結果セット  
を返すことができます。
- 計算ローのデータを処理するために、Open Server アプリケーショ  
ンが行うことを次に示します。
  - a 計算ロー識別子を定義するために、`srv_alt_header` を呼び出し  
ます。
  - b クライアントが受け取る時のカラム・データのフォーマッ  
トを記述するために、各カラムで `srv_alt_descfmt` を呼び出し  
ます。
  - c データをローカル・プログラム変数にバインドするために、  
各カラムで `srv_alt_bind` を呼び出します。
  - d 計算ローの各カラムが記述され、そのデータがプログラム変  
数にバインドされると、ローをクライアントに送信するた  
めに、`srv_alt_xferdata` を呼び出します。

- *varaddrp*、*lenp*、および *indp* が指すバッファの内容は、*srv\_xferdata* が呼び出されるまで有効である必要はありません。

参照

[srv\\_alt\\_descfmt](#), [srv\\_alt\\_header](#), [srv\\_alt\\_xferdata](#), 「CS\_DATAFMT 構造体」(57 ページ)

## srv\_alt\_descfmt

説明

計算ロー・カラムの集合演算子と、クライアントに返されるカラム・データのフォーマットを記述します。

構文

```
CS_RETCODE srv_alt_descfmt(spp, altid, optype,
                           operand, item, clfmtp)
```

```
SRV_PROC      *spp;
CS_INT        altid;
CS_INT        optype;
CS_TINYINT    operand;
CS_INT        item;
CS_DATAFMT    *clfmtp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*altid*

この計算カラムが含まれている計算ロー用のユニークな識別子です。*altid* は、*srv\_alt\_header* を使って定義されます。

*item*

計算ロー内のカラムの番号です。カラム番号は、1 から始まります。

*optype*

計算ロー・カラムの集合演算子のタイプです。次の表に、有効な演算子のタイプを示します。

**表 3-5 : optype の値 (srv\_alt\_descfmt)**

演算子の種類	機能
CS_OP_COUNT	カウント集合演算子
CS_OP_SUM	合計集合演算子
CS_OP_AVG	平均集合演算子
CS_OP_MIN	最小集合演算子
CS_OP_MAX	最大集合演算子

*operand*

集合演算の対象となる select リストのカラムです。

*clfmtp*

CS\_DATAFMT 構造体へのポインタです。この構造体では、クライアントがカラム・データを受け取るときに使用する、カラム・データを含んだフォーマットを記述します。

戻り値

**表 3-6 : 戻り値 (srv\_alt\_descfmt)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE      ex_srv_alt_descfmt PROTOTYPE((
SRV_PROC        *sproc,
CS_INT          altid,
CS_DATAFMT      clfmtp[]
));
/*
** EX_SRV_ALT_DESCFMT
**   An example routine to describe the aggregate operator of 2
**   compute row columns and the format of each of the two column
**   data returned to the client. We will do the sum on the first
**   column and average on the second column.
**
** Arguments:
**   sproc      A pointer to an internal thread control structure.
**   altid      The id for the compute row in which this compute
**              column is contained. The altid is obtained by
**              calling srv_alt_header.
**   clfmtp     A pointer to the array of structures describing
```



```

**          the format of the compute row column
**          data when the client receives it.
**
** Returns:
**   CS_SUCCEEDED   If the aggregate operator and the datatype of
**                   the compute row columns were successfully
**                   described.
**   CS_FAIL        An error was detected.
*/
CS_RETCODE      ex_srv_alt_descfmt(sproc, altid, clfmtmp)
SRV_PROC        *sproc;
CS_INT          altid;
CS_DATAFMT      clfmtmp[];
{
    /*
    ** Describe the aggregate operator of the first compute row
    ** column and the format of the column data.
    */
    if ( srv_alt_descfmt(sproc, altid, (CS_INT)1, CS_OP_SUM,
        (CS_TINYINT)1, &clfmtmp[0]) == CS_FAIL )
    {
        return(CS_FAIL);
    }
    /*
    ** Now do the same for the second column if (srv_alt_descfmt
    ** (sproc, altid, (CS_INT)2, CS_OP_AVG, (CS_TINYINT)2,
    ** &clfmtmp[1]) == CS_FAIL )
    {
        return(CS_FAIL);
    }
    */

    return (CS_SUCCEEDED);
}

```

#### 使用法

- 計算ローの情報を返す Adaptive Server Enterprise の機能を模擬的に  
行うアプリケーションのみが、`srv_alt_descfmt` を呼び出す必要があ  
ります。`srv_alt_descfmt` は、Adaptive Server Enterprise へのゲート  
ウェイとしての役割を果たしているアプリケーションにとって便  
利です。
- `srv_alt_descfmt` は、アプリケーションがクライアントに送る計算  
ロー・カラムを記述します。アプリケーションは、計算ローの各  
カラムで、一度ずつ `srv_alt_descfmt` を呼び出します。

- `srv_alt_descfmt` ルーチンは、次の表に示す `CS_DATAFMT` フィールドに対して、読み取り (`CS_GET`) や設定 (`CS_SET`) を行います。他のすべてのフィールドは、`srv_alt_descfmt` には未定義です (“`clfmt`” は構造体を指すポインタであることに注意してください)。

**表 3-7 : 使用される CS\_DATAFMT 構造体フィールド (srv\_alt\_descfmt)**

フィールド	CS_SET	CS_GET
<code>clfmt</code> → <code>namelen</code>	名前の長さ	名前の長さ
<code>clfmt</code> → <code>status</code>	パラメータ/カラム・ステータス	パラメータ・ステータス
<code>clfmt</code> → <code>name</code>	パラメータ/カラム名	パラメータ名
<code>clfmt</code> → <code>datatype</code>	リモート・データ型をここに設定	リモート・データ型をここから取得
<code>clfmt</code> → <code>maxlength</code>	リモート・データ型の最大長をここに設定	リモート・データ型の最大長をここから取得
<code>clfmt</code> → <code>format</code>	リモート・データ型のフォーマット	リモート・データ型のフォーマット

- `clfmt` により記述されるフォーマットが、その後で `srv_alt_bind` (`osfmt`) で設定されたアプリケーション・プログラム変数のフォーマットと異なっている場合には、Open Server は、自動的にデータを `clfmt` フォーマット記述に変換します。
- 計算ローのデータを処理するために、Open Server アプリケーションが行うことを次に示します。
  - a 計算ロー識別子を定義するために、`srv_alt_header` を呼び出します。
  - b クライアントが受け取るときのカラム・データのフォーマットを記述するために、各カラムで `srv_alt_descfmt` を呼び出します。
  - c データをローカル・プログラム変数にバインドするために、各カラムで `srv_alt_bind` を呼び出します。
  - d 計算ローの各カラムが記述され、そのデータがプログラム変数にバインドされると、ローをクライアントに送信するために、`srv_alt_xferdata` を呼び出します。

参照

[srv\\_alt\\_bind, srv\\_alt\\_header, srv\\_alt\\_xferdata, 「CS\\_DATAFMT 構造体」\(57 ページ\)](#)

## srv\_alt\_header

**説明** 計算ローのロー識別子および *bylist* を記述します。

**構文** CS\_RETCODE *srv\_alt\_header*(*spp*, *altid*, *numbylist*,  
*bylistarrayp*)

```
SRV_PROC      *spp;
CS_INT        altid;
CS_INT        numbylist;
CS_SMALLINT   *bylistarrayp;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*altid*

この計算ロー用のユニークな識別子です。

*numbylist*

計算ローの *bylist* 内のカラム数です。

*bylistarrayp*

計算ロー用の *bylist* を構成しているカラム番号の配列を指すポインタです。要素は、*numbylist* に指定されている数だけあります。*numbylist* が 0 の場合は、*bylistarrayp* は無視されます。

**戻り値**

**表 3-8 : 戻り値 (srv\_alt\_header)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include      <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE    ex_srv_alt_header PROTOTYPE((
SRV_PROC      *spp
));
/*
** EX_SRV_ALT_HEADER
**
** Example routine to illustrate the use of srv_alt_header
** to describe a compute row's row identifier and bylist.
**
** Arguments:
** spp - A pointer to an internal thread control structure.
```

```

**
** Returns:
**
**     CS_SUCCEED      A compute row was successfully described.
**     CS_FAIL        An error was detected.
*/
CS_RETCODE      ex_srv_alt_header(spp)
SRV_PROC        *spp;
{
    CS_INT        altid;
    CS_SMALLINT  bylist[2];

    /*
    ** Let us describe a fictitious compute row with altid =1,
    ** and bylist = [2,4].
    */
    altid = (CS_INT)1;
    bylist[0] = (CS_SMALLINT)2;
    bylist[1] = (CS_SMALLINT)4;

    if (srv_alt_header(spp, altid,
        sizeof(bylist)/sizeof(CS_SMALLINT),
        bylist) == CS_FAIL)
        return(CS_FAIL);

    return (CS_SUCCEED);
}

```

#### 使用法

- 計算ローの情報を返す Adaptive Server Enterprise の機能を模擬的に  
行うアプリケーションのみが、`srv_alt_header` を呼び出す必要があり  
ます。`srv_alt_header` は、Adaptive Server Enterprise へのゲート  
ウェイとしての役割を果たしているアプリケーションにとって便利  
です。
- `srv_alt_header` は、各計算ローにユニークな識別子を割り当て、各  
計算ローと関連する `bylist` を記述します。各計算ローで一度ずつ  
`srv_alt_header` を呼び出さなければなりません。
- Adaptive Server Enterprise では、計算ローは、Transact-SQL `select` 文  
の `compute` 句によって生成されます。Transact-SQL の `select` 文に複  
数の `compute` 句がある場合には、各句によって別々の計算ローが  
生成されます。Open Server は、Adaptive Server Enterprise の  
Transact-SQL の `compute` 句に対する応答を真似て、計算データの  
ローを返すことができます。

- Transact-SQL の `select` 文の `compute` 句には、`by` というキーワードを含めて、その後にはカラムのリストを続けることができます。このリストは、"`bylist`" として知られ、指定されたカラムの値の変更に基づいて、結果をサブグループに分類します。`compute` 句の集合演算子は各サブグループに適用され、各サブグループで計算ローが生成されます。
- `*bylistarray` の配列では、`bylist` の各カラムに関連する番号が保管されます。この番号は、`select` 文内のカラムの位置によって決まります。たとえば、カラムが、`select` 文での3つ目のアイテムである場合には、配列でナンバー3としてリストされます。
- 計算ローのデータを処理するために、Open Server アプリケーションが行うことを次に示します。
  - a 計算ロー識別子を定義するために、`srv_alt_header` を呼び出します。
  - b クライアントが受け取る時のカラム・データのフォーマットを記述するために、各カラムで `srv_alt_descfmt` を呼び出します。
  - c データをローカル・プログラム変数にバインドするために、各カラムで `srv_alt_bind` を呼び出します。
  - d 計算ローの各カラムが記述され、そのデータがプログラム変数にバインドされると、ローをクライアントに送信するために、`srv_alt_xferdata` を呼び出します。

参照

[srv\\_alt\\_bind](#), [srv\\_alt\\_descfmt](#), [srv\\_alt\\_xferdata](#)

## srv\_alt\_xferdata

説明 クライアントに計算ローを送ります。

構文 CS\_RETCODE `srv_alt_xferdata`(`spp`, `altid`)

SRV\_PROC \*`spp`;

CS\_INT `altid`;

パラメータ

`spp`

内部スレッド制御構造体へのポインタです。

*altid*

クライアントに送られる計算ロー用のユニークな識別子です。*altid* は、*srv\_alt\_header* を使って定義されます。

戻り値

**表 3-9 : 戻り値 (srv\_alt\_xferdata)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local Prototype.
*/

CS_RETCODE      ex_srv_alt_xferdata PROTOTYPE((
SRV_PROC        *spp,
CS_INT          altid
));

/*
** EX_SRV_ALTXFERDATA
**
** Example routine to send a compute row the the client using
** srv_altxferdata.
**
** Arguments:
** spp          A pointer to an internal thread control structure.
** altid       The compute row identifier (defined using
**             srv_alt_header).
**
** Returns:
**
** CS_SUCCEED  The row was sent to the client.
** CS_FAIL     An error was detected.
*/
CS_RETCODE      ex_srv_alt_xferdata(spp, altid)
SRV_PROC        *spp;
CS_INT          altid;
{
    /*
    ** Send the compute row to the client.
    */
    if (srv_alt_xferdata(spp, altid) != CS_SUCCEED)
    {
```

```

        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

#### 使用法

- 計算ローの情報を返す **Adaptive Server Enterprise** の機能を模倣的に  
行うアプリケーションのみが、`srv_alt_xferdata` を呼び出す必要があ  
ります。これは、**Adaptive Server Enterprise** へのゲートウェイとし  
ての役割を果たしているアプリケーションにとって便利です。
- `srv_alt_xferdata` は、クライアントに計算ローを送ります。この  
ルーチンは、各 *altid* で一度呼び出されます。
- 計算ローのデータを処理するために、**Open Server** アプリケーショ  
ンが行うことを次に示します。
  - a 計算ロー識別子を定義するために、`srv_alt_header` を呼び出し  
ます。
  - b クライアントが受け取る時のカラム・データのフォーマッ  
トを記述するために、各カラムで `srv_alt_descfmt` を呼び出しま  
す。
  - c データをローカル・プログラム変数にバインドするために、  
各カラムで `srv_alt_bind` を呼び出します。
  - d 計算ローの各カラムが記述され、そのデータがプログラム変  
数にバインドされると、ローをクライアントに送信するた  
めに、`srv_alt_xferdata` を呼び出します。
- `srv_senddone` で送信完了のステータスを送る前に、すべての計算  
ローをクライアントに送ります。

#### 参照

[srv\\_alt\\_bind](#), [srv\\_alt\\_header](#), [srv\\_alt\\_descfmt](#)

## srv\_bind

#### 説明

カラムまたはパラメータにプログラム変数を記述し、バインドします。

#### 構文

```
CS_RETCODE srv_bind(spp, cmd, type, item, osfmp,
                    varaddrp, varlenp, indp)
```

```
SRV_PROC    *spp;
CS_INT      cmd;
CS_INT      type;
CS_INT      item;
```

```

CS_DATAFMT  *osfmt;
CS_BYTE     *varaddrp;
CS_INT      *varlenp;
CS_SMALLINT *indp;
    
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

*cmd* は、プログラム変数が、クライアントに送られるデータまたはクライアントから入ってくるデータを保存するかどうかを示します。次の表に、*cmd* の有効値を示します。

**表 3-10 : *cmd* の値 (srv\_bind)**

値	説明
CS_SET	* <i>varaddrp</i> のデータは、 <i>srv_xferdata</i> が呼び出されたときにクライアントに送信される。
CS_GET	* <i>varaddrp</i> は、 <i>srv_xferdata</i> の呼び出しの後に、クライアントからのデータで初期化される。

*type*

プログラム変数に保管されたり、プログラム変数から読み取られるデータの型です。表 3-11 に、*type* の有効値を示します。

**表 3-11 : *type* の値 (srv\_bind)**

Type	有効な cmd	データの内容
SRV_RPCDATA	CS_SET または CS_GET	RPC またはストアド・プロシージャ・パラメータ
SRV_ROWDATA	CS_SET のみ	結果ロー・カラム
SRV_CURDATA	CS_GET のみ	カーソル・パラメータ
SRV_KEYDATA	CS_GET のみ	カーソル・キー・カラム
SRV_ERRORDATA	CS_SET のみ	エラー・メッセージ・パラメータ
SRV_DYNAMICDATA	CS_SET または CS_GET	動的 SQL パラメータ
SRV_NEGDATA	CS_SET または CS_GET	ネゴシエーション・ログイン・パラメータ
SRV_MSGDATA	CS_SET または CS_GET	メッセージ・パラメータ
SRV_LANGDATA	CS_GET のみ	言語パラメータ

*item*

カラムまたはパラメータの番号です。カラムやパラメータ番号は、1 から始まります。



*osfntp*

*CS\_DATAFMT* 構造体を指すポインタです。この構造体は、*\*varaddrp* に保存されているデータのフォーマットを記述します。

*varaddrp*

カラムまたはパラメータのデータがバインドされているプログラム変数へのポインタです。

*varlenp*

*varaddrp* の長さを指すポインタです。その正確な意味およびプロパティは、*cmd* の値によって変わります。表 3-12 に、*varlenp* の有効値を示します。

表 3-12 : *varlenp* の値 (*srv\_bind*)

cmd	varlenp
<i>CS_SET</i> (クライアントに送られるデータ)	<ul style="list-style-type: none"> <li>• <i>NULL</i> は使用不可。</li> <li>• <i>*varaddrp</i> のデータの実際の長さを指す。</li> <li>• <i>srv_xferdata</i> が呼び出されるまで有効である必要はない。</li> </ul>
<i>CS_GET</i> (クライアントから入ってくるデータ)	<ul style="list-style-type: none"> <li>• <i>NULL</i> が使用可能 (<i>Open Server</i> アプリケーションがすでにデータ長を認識していることを示す)。</li> <li>• <i>Open Server</i> が実データ長を格納するプログラム変数へのポインタ。</li> <li>• <i>srv_xferdata</i> への呼び出しの後に値が取り込まれる。</li> </ul>

データを取得するときには、アプリケーションが *srv\_xferdata* を呼び出すまで、*\*varlenp* は空です。*srv\_xferdata* を呼び出すと、*Open Server* は新しく受信した値の長さをバッファに格納します。データを送信するときには、アプリケーションは、データを送信するために *srv\_xferdata* を呼び出す前に *\*varlenp* が指すバッファにデータを設定します。

*indp*

*null* 値のインジケータが格納されているバッファへのポインタです。表 3-13 に、*\*indp* の有効値を示します。

表 3-13 : *indp* の値 (*srv\_bind*)

値	意味
<i>CS_NULLDATA</i>	カラムまたはパラメータのデータは <i>null</i> である。
<i>CS_GOODDATA</i>	カラムまたはパラメータのデータは <i>null</i> ではない。

*indp* が *NULL* の場合、カラム・データは有効、つまり、*NULL* ではないとみなされます。

戻り値

**表 3-14 : 戻り値 (srv\_bind)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
CS_RETCODE      ex_srv_bind PROTOTYPE((
SRV_PROC        *spp,
CS_INT          *nump,
CS_BYTE         *namep,
CS_INT          *lenp
));
/*
** EX_SRV_BIND
**
** Example routine using srv_bind to describe and bind two
** program.
** variables to receive client RPC parameters.For this
** example, the
** RPC is passed an employee number, and last name.A third
** program.
** variable will be bound to receive the length of the
** employee' s name.
** This routine is called prior to srv_xferdata, which will
** actually transfer the data into the program variables.
**
** Arguments:
** spp      A pointer to an internal thread control structure.
** nump     A Pointer to the integer to receive the employee
** number.
** namep    A Pointer to the memory area to receive the
** employee name.
** lenp     A Pointer to the integer to receive the length of
** the employee' s name.(On input, points to the
** maximum length of the memory area available.)
**
** Returns:
** CS_SUCCEED   Program variables were successfully bound.
** CS_FAIL      An error was detected.
**/
CS_RETCODE      ex_srv_bind(spp, nump, namep, lenp)
```

```

SRV_PROC                *spp;
CS_INT                  *nump;
CS_BYTE                 *namep;
CS_INT                  *lenp;
{
    CS_INT                param_no;
    CS_DATAFMT            varfmt;
    srv_bzero((CS_VOID *)&varfmt, (CS_INT)sizeof(varfmt));
    /*
    ** First, bind the integer to receive the employee number,
    ** param 1. Here, we know the length of the data, so no
    ** length pointer is required.
    */
    param_no = 1;
    varfmt.datatype = CS_INT_TYPE;
    varfmt.maxlength = (CS_INT)sizeof(CS_INT);
    if (srv_bind(spp, (CS_INT)CS_GET, (CS_INT)SRV_RPCDATA,
                param_no, &varfmt, (CS_BYTE *)nump, (CS_INT *)NULL,
                (CS_SMALLINT *)NULL) != CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    /*
    ** Then, bind the character memory to receive the
    ** employee name, param 2.
    */
    param_no = 2;
    varfmt.datatype = CS_CHAR_TYPE;
    varfmt.maxlength = *lenp;
    if (srv_bind(spp, (CS_INT)CS_GET, (CS_INT)SRV_RPCDATA,
                param_no,
                &varfmt, namep, lenp, (CS_SMALLINT *)NULL) !=
        CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

#### 使用法

- `srv_bind` は、ロー・カラムまたはパラメータのフォーマットを記述し、それをアプリケーション・プログラム変数と関連付けます。
- `srv_bind` は、結果ローの各カラムでまたはパラメータ・ストリームの各パラメータで、一度ずつ呼び出されなければなりません。

- ローを送信している間にローカル・プログラム変数アドレス (*varaddrp*、*varlenp*、または *indp*) を変えたいアプリケーションでは、変更を行うたびに *srv\_bind*、続いて *srv\_xferdata* を呼び出されなければなりません。
- **Server-Library** アプリケーションは、クライアントにデータを 2 段階に分けて送信します。

最初に、*srv\_bind* を **CS\_SET** と等しい *cmd* で呼び出します。*varaddrp*、*varlenp*、*indp* の各パラメータはそれぞれ、検索するデータへのポインタ、長さへのポインタ、インジケータ変数へのポインタを保持しています。このとき、**Server-Library** はこれらのポインタ・パラメータに渡されたアドレスを記録します。

これらの値は、アプリケーションが *srv\_xferdata* を呼び出すまで、つまり **Server-Library** がそれらのメモリ・ロケーションから値を読み取るまで有効でなければなりません。たとえば、複数のデータ・アイテムが *srv\_bind* への異なる呼び出しに対して渡されるとときには、異なるバッファを使用する必要があります。

- エラー・データ・パラメータは、*srv\_sendinfo* の呼び出しの直後、*srv\_senddone* を呼び出す前に記述され (*srv\_descfmt*)、バインドされ (*srv\_bind*)、クライアントに送られ (*srv\_xferdata*) なければなりません。ルーチン *srv\_descfmt*、*srv\_bind*、および *srv\_xferdata* の *type* 引数は **SRV\_ERRORDATA** に設定されています。
- メッセージ・データ・パラメータは、*srv\_msg* ルーチンの呼び出しの後に記述され (*srv\_descfmt*)、バインドされ (*srv\_bind*)、転送 (*srv\_xferdata*) されなければなりません。ルーチン *srv\_descfmt*、*srv\_bind*、および *srv\_xferdata* の *type* 引数は **SRV\_MSGDATA** に設定されています。
- *srv\_bind* ルーチンは、次の表に示す **CS\_DATAFMT** フィールドに対して、読み取り (**CS\_GET**) や設定 (**CS\_SET**) を行います。他のすべてのフィールドは、*srv\_bind* には未定義です (“*osfmt*” は構造体を指すポインタであることに注意してください)。

表 3-15 : 使用される CS\_DATAFMT フィールド (srv\_bind)

フィールド	CS_SET オペレーションにおける定義	CS_GET オペレーションにおける定義
<i>osfmp</i> → <i>datatype</i>	アプリケーション・プログラム変数のデータ型	アプリケーション・プログラム変数のデータ型
<i>osfmp</i> → <i>maxlength</i>	プログラム変数の実際の長さ	プログラム変数の最大長
<i>osfmp</i> → <i>count</i>	0 または 1	0 または 1
<i>osfmp</i> → <i>status</i>	null 値を送信している場合、CS_CANBENULL を設定	未使用

カラムの null 値を送信するには、そのカラムの CS\_DATAFMT 構造体の *status* 値に CS\_CANBENULL ビット・セットが設定されている必要があります。CS\_DATAFMT 構造体の *status* の有効値については、表 2-9 (60 ページ) を参照してください。

- *osfmp* によって記述されたフォーマットがクライアントから受信したデータのフォーマット (*cmd* を CS\_GET に設定) と異なる場合、Open Server はデータを自動的に *osfmp* に変換します。フォーマットが、クライアントに送られるフォーマット (*cmd* を CS\_SET に設定) と異なる場合には、Open Server は、自動的にフォーマットをクライアント・フォーマットに変換します (*clfmp*)。

## 参照

[srv\\_cursor\\_props](#), [srv\\_descfmt](#), [srv\\_msg](#), [srv\\_sendinfo](#), [srv\\_xferdata](#), 「CS\_DATAFMT 構造体」(57 ページ), 「パラメータとロー・データの処理」(152 ページ)。

## srv\_bmove

## 説明

バイトを、1つのメモリ・ロケーションから別のメモリ・ロケーションにコピーします。

## 構文

```
CS_VOID srv_bmove(sourcep, destp, count)
```

```
CS_VOID *sourcep;
CS_VOID *destp;
CS_INT count;
```

## パラメータ

*sourcep*

コピーされるデータの元を指す非 null ポインタです。

*destp*

コピーされるデータの行き先を指す非 null ポインタです。

*count*

*sourcep* から *destp* にコピーされるバイト数です。

戻り値

なし。

例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
CS_VOID          ex_srv_bmove PROTOTYPE((
CS_VOID          *src,
CS_VOID          *dest,
CS_INT           count
));

/*
** EX_SRV_BMOVE
**
**      Example routine to copy data from one area of memory to
**      another.
**
** Arguments:
**   src    - The address of the source data.
**   dest   - The address of the destination buffer.
**   count  - The number of bytes to copy.
**
** Returns:
**      Nothing.
**/
CS_VOID          ex_srv_bmove(src, dest, count)
CS_VOID          *src;
CS_VOID          *dest;
CS_INT           count;
{
    /*
    ** Call the Open Server routine that will do the
    ** actual copy.
    */
    srv_bmove(src, dest, count);

    /*
    ** All done.
    */
}
```

```

    return;
}

```

#### 使用法

- `srv_bmove` は、`count` バイトをメモリ・ロケーション `*sourcep` からメモリ・ロケーション `*destp` にコピーします。
- `sourcep` も `destp` も、有効な非 `null` ポインタでなければ、メモリ・フォールトが生成されます。
- 移動するのは、`count` バイトだけで、`null` ターミネータは追加されません。

#### 参照

[srv\\_bzero](#)

## srv\_bzero

#### 説明

メモリ・ロケーションの内容を 0 に設定します。

#### 構文

```
CS_VOID srv_bzero(locationp, count)
```

```
CS_VOID *locationp;
CS_INT count;
```

#### パラメータ

*locationp*

0 に設定されるバッファのアドレスを指す非 `null` ポインタです。

*count*

*locationp* で、0x00 の値に設定するバイト数です。

#### 戻り値

なし。

#### 例

```

#include          <ospublic.h>
/*
** Local Prototype
**/
CS_RETCODE      ex_srv_bzero PROTOTYPE((
    CS_VOID      *locationp,
    CS_INT       cnt
))
/*
** EX_SRV_BZERO
** Example routine to set the contents of a section of memory
** to zero using srv_bzero
**
** Arguments:

```

```

**
**      memp          Pointer to section of memory.
**      count        Number of bytes to set to zero.
**
** Returns
**      CS_SUCCEEDED Arguments were valid and srv_bzero called.
**      CS_FAIL       An error was detected.
**
*/
CS_RETCODE      ex_srv_bzero(memp, count)
CS_VOID         *memp;
CS_INT         count;
{
    /* Check arguments.*/
    if(memp == (CS_VOID *)NULL)
    {
        return(CS_FAIL);
    }
    if(count < 0)
    {
        return(CS_FAIL);
    }

    /*
    ** Set the section of memory to the value 0x00.
    */
    (CS_VOID)srv_bzero(memp, count);
    return (CS_SUCCEEDED);
}

```

使用法

- `srv_bzero` は、*locationp* のメモリ・ロケーションで *count* 数のバイトを `0x00` の値に設定します。
- *locationp* が有効な非 `null` ポインタでない場合は、メモリ・フォールトが発生します。

参照

[srv\\_bmove](#)

## srv\_callback

説明

スレッドに対して状態遷移ハンドラをインストールします。

構文

`CS_RETCODE srv_callback(spp, callback_type, funcp)`



```
SRV_PROC      *spp;
CS_INT        callback_type;
CS_RETCODE    (*funcp)();
```

## パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*callback\_type*

コールバックがインストールされている対象の状態遷移を示す整数です。表 3-16 に、*callback\_type* の有効値を示します。

表 3-16 : *callback\_type* の値 (*srv\_callback*)

値	説明
SRV_C_EXIT	スレッドは、 <i>srv_spawn</i> で指定されたエントリ・ポインタから戻ったか、あるいは切断されたクライアントと関連している。ハンドラは、終了しているスレッドのコンテキストで実行。
SRV_C_PROCEXEC	レジスタード・プロシージャが呼び出され、実行しようとしている。ハンドラは、レジスタード・プロシージャを要求したスレッドのコンテキストで実行する。
SRV_C_RESUME	スレッドは再開中である。ハンドラはスケジューラ・スレッドのコンテキストで実行され、そのスタックを使用。
SRV_C_SUSPEND	スレッドは中断している。ハンドラは、中断しているスレッドのコンテキストで実行され、そのスタックを使用する。
SRV_C_TIMESLICE	この状態遷移のためにインストールするコールバック・ルーチンは、SRV_S_TIMESLICE、SRV_S_VIRTCLKRATE、SRV_S_VIRTTIMER のサーバ・プロパティによって決められた期間 (タイム・スライス) を、スレッドが実行したときに呼び出される。「 <a href="#">srv_props</a> 」 (377 ページ) および「 <a href="#">プロパティ</a> 」 (157 ページ) を参照してください。

*funcp*

指定の状態遷移が起こったときに呼び出す関数へのポインタです。

コールバック関数は、スレッド・ポインタ引数を持ちます。

戻り値

**表 3-17 : 戻り値 (srv\_callback)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <stdio.h>
#include      <ospublic.h>
/*
** Local Prototye
**/
CS_RETCODE      suspend_handler PROTOTYPE((
SRV_PROC        *srvproc
));
CS_RETCODE ex_srv_callback PROTOTYPE((
SRV_PROC        *srvproc
));

CS_RETCODE      suspend_handler(srvproc)
SRV_PROC        *srvproc;
{
    printf( "Wake me when it' s over...¥n" );
    return (CS_SUCCEED);
}

/*
** EX_SRV_CALLBACK
**
**      Example routine to install a state transition handler.
**
** Arguments:
**      srvpro - A pointer to an internal thread control structure.
**
** Returns:
**
**      CS_SUCCEED
**      CS_FAIL
**/
CS_RETCODE      ex_srv_callback(srvproc)
SRV_PROC        *srvproc;
{
    return(srv_callback(srvproc, SRV_C_SUSPEND,
                        suspend_handler));
}
```

## 使用法

- スレッドのステータスが別のステータスに遷移するときに実行するルーチンを、`srv_callback` を使って指定します。
- アプリケーションは、ステータス遷移中のスレッドへのポインタを指定して、コールバック・ルーチンを呼び出します。
- 表 3-18 に、各コールバック・ルーチンが返す値を示します。

表 3-18 : コールバック・ルーチンの有効な戻り値 (`srv_callback`)

コールバック・ルーチンのタイプ	戻り値	戻り値の説明
SRV_C_EXIT	Open Server では無視されるが、将来の互換性を考えて SRV_CONTINUE に設定することを推奨。	
SRV_C_PROCEXEC	SRV_S_INHIBIT	レジスタード・プロセスの実行を中止。
	SRV_S_CONTINUE	レジスタード・プロセスの実行を継続。
SRV_C_RESUME	Open Server では無視されるが、将来の互換性を考えて SRV_CONTINUE に設定することを推奨。	
SRV_C_SUSPEND	Open Server では無視されるが、将来の互換性を考えて SRV_CONTINUE に設定することを推奨。	
SRV_C_TIMESLICE	SRV_CONTINUE	実行を中断せず継続。
	SRV_TERMINATE	スレッドを終了。
	SRV_DEBUG	後でデバッガで検査するために、スレッドをデバッグ・キューに追加。

- コールバックの中には、プラットフォームによっては使用できないものもあります。現在のプラットフォームでハンドラがインストール可能かどうかは、`srv_capability` を呼び出して調べることができます。

- 以前の `srv_callback` への呼び出しでインストールしたコールバック・ルーチンを削除するには、そのコールバック・ルーチンの代わりに `null` 関数をインストールします。たとえば、先にインストールした `SRV_C_TIMESLICE` ハンドラのインストールを取り消したい場合は、次のコマンドを発行します。

```
srv_callback(spp, SRV_C_TIMESLICE, NULL);
```

- アプリケーションが、コールバック・ハンドラを通知にだけしか使用しない場合は、`funcp` 引数を `NULL` に設定してください。詳細については、「[レジスタード・プロシージャ](#)」(185 ページ) を参照してください。

参照

[srv\\_capability](#), [srv\\_props](#), [srv\\_termproc](#)

## srv\_capability

説明

Open Server がプラットフォーム依存のサービスをサポートするかどうかを決定します。

構文

```
CS_BOOL srv_capability(capability)
CS_INT      capability;
```

パラメータ

*capability*

テストする Open Server サービスを表す定数です。表 3-19 に *capability* の有効値を示します。

表 3-19 : capability の値 (srv\_capability)

値	説明
SRV_C_DEBUG	srv_dbg_stack および srv_dbg_switch をサポートする。
SRV_C_EXIT	スレッドが終了すると、コールバック・ルーチンを呼び出すことができる。
SRV_C_RESUME	スレッドが実行を再開すると、コールバック・ルーチンを呼び出すことができる。
SRV_C_PREEMPT	プリエンティブ・スケジューリングをサポートする。
SRV_C_SELECT	srv_select をサポートする。
SRV_C_SUSPEND	スレッドが中断すると、コールバック・ルーチンを呼び出すことができる。
SRV_C_TIMESLICE	スレッドがクロック・チックの最大数を超えると、コールバック・ルーチンを呼び出すことができる。
SRV_POLL	srv_poll をサポートする。

戻り値

表 3-20 : 戻り値 (srv\_capability)

戻り値	意味
CS_TRUE	Open Server はサービスをサポートする。
CS_FALSE	Open Server はサービスをサポートしない。

例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
extern CS_RETCODE ex_srv_capability PROTOTYPE((void));
/*
** EX_SRV_CAPABILITY
**
** Example routine to determine whether srv_poll is supported
** on this platform.
**
** Arguments:
** None.
**
** Returns:
**
** CS_SUCCEED   srv_poll is supported on this platform.
** CS_FAIL      srv_poll is not supported on this platform.
**/
CS_RETCODE      ex_srv_capability()
{
```

```

CS_BOOL supported;
/*
** Check to see whether srv_poll is supported on this
** platform.
*/
supported = srv_capability(SRV_C_POLL);
/*
** If "supported" is CS_TRUE, we return CS_SUCCEED, if it is
** CS_FALSE we return CS_FAIL.
*/
return(supported ?CS_SUCCEED :CS_FAIL);
}

```

使用法

- `srv_capability` を使用すると、移植可能な Open Server アプリケーションを作成しながら、プラットフォームによっては使用できないサービスも利用することができます。
- Open Server には、プラットフォーム機能とプロトコル機能の2つの機能があります。`srv_capability` ルーチンは、プラットフォーム機能に関するものです。`srv_capability_info` ルーチンは、プロトコル機能に関するものです。詳細については、`srv_capability_info` のページを参照してください。

参照

[srv\\_callback](#), [srv\\_capability](#), [srv\\_dbg\\_stack](#), [srv\\_dbg\\_switch](#), [srv\\_poll](#) (UNIXのみ), [srv\\_select](#) (UNIXのみ), [srv\\_capability\\_info](#)

## srv\_capability\_info

説明

クライアント接続に関する機能情報を定義または取得します。

構文

```

CS_RETCODE srv_capability_info(spp, cmd, type,
                               capability, valp)
SRV_PROC      *spp;
CS_INT        cmd;
CS_INT        type;
CS_INT        capability;
CS_VOID       *valp;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*cmd*  
Open Server アプリケーションが機能情報を定義しているのか取得しているのかを示します。表 3-21 に、*cmd* の有効値を示します。

表 3-21 : cmd の値 (srv\_capability\_info)

値	意味
CS_SET	Open Server アプリケーションは機能情報を定義している。
CS_GET	Open Server アプリケーションは、クライアントからの機能情報を取得している。

*type*

機能グループのタイプです。表 3-22 に、有効な 2 つのタイプを示します。

表 3-22 : type の値 (srv\_capability\_info)

値	意味
CS_CAP_REQUEST	クライアントが送る可能性のあるコマンド。
CS_CAP_RESPONSE	クライアントが Open Server アプリケーションを保留させる可能性のある応答。

*capability*

対象となる機能項目を指定します。type カテゴリのすべての機能項目のビットマップを取得するには、*capability* を CS\_ALL\_CAPS に設定します。すべての要求および応答機能のリストについては、「機能」(25 ページ) を参照してください。

*valp*

プログラム変数へのポインタです。クライアントに情報を送る場合 (CS\_SET)、この変数には機能の値が設定されます。クライアントからの情報を取得する場合 (CS\_GET)、Open Server によって、この変数に機能の値が設定されます。*valp* は、アプリケーションが個々の機能アイテムを定義または取得している場合には、CS\_BOOL ポインタ、およびすべての機能項目のビットマップを定義または取得している場合には、CS\_CAP\_TYPE ポインタでなくてはなりません (つまり、*capability* は CS\_ALL\_CAPS)。

## 戻り値

表 3-23 : 戻り値 (srv\_capability\_info)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <ospublic.h>
CS_RETCODE ex_srv_capability_info PROTOTYPE((
    SRV_PROC *spp
));
/*
```

```

** EX_SRV_CAPABILITY_INFO
**
**      Example routine to retrieve and define capability
**      information on a client connection.
**
**      This routine must called in the context of the connect
**      handler, so that it is legal to negotiate capabilities.
**
** Arguments:
**      spp      A pointer to an internal thread control structure.
**
** Returns:
**      CS_SUCCEED - Successfully retrieved and bound capability
**      information.
**      CS_FAIL    - An error was detected.
**
*/
CS_RETCODE      ex_srv_capability_info(spp)
SRV_PROC        *spp;

{
    CS_RETCODE      retval; /* Return value from Open */
                    /* Server API calls.*/

    CS_CAP_TYPE     capabilities; /* Our bit mask.*/

    CS_BOOL         value; /* Set to CS_TRUE or CS_FALSE */
                    /* for individual capabilities.*/

    /*
    ** In this example, we don't want to support text or image,
    ** so we'll see first if the client has requested this.
    ** We'll do this by getting the entire bit mask.
    */
    retval = srv_capability_info(spp, CS_GET, CS_CAP_REQUEST,
                                CS_ALL_CAPS, (CS_VOID *)&capabilities);

    if (retval == CS_FAIL)

    {
        return(CS_FAIL);
    }

    /*
    ** Turn off text and image.
    **

```



```

** The other way to do this is to just clear the
** CS_DATA_TEXT and CS_DATA_IMAGE bits in the capabilities
** bit mask, and then call srv_capability_info() with
** CS_ALL_CAPS for the "type" parameter and the altered
** bit mask as the value.
*/
if (CS_TST_CAPMASK(&capabilities, CS_DATA_TEXT) == CS_TRUE)
{
    value = CS_FALSE;
    retval = srv_capability_info(spp, CS_SET,
        CS_CAP_REQUEST, CS_DATA_TEXT, (CS_VOID *)&value);
    if (retval == CS_FAIL)
    {
        return(CS_FAIL);
    }
}

if (CS_TST_CAPMASK(&capabilities, CS_DATA_IMAGE) == CS_TRUE)
{
    value = CS_FALSE;
    retval = srv_capability_info(spp, CS_SET,
        CS_CAP_REQUEST, CS_DATA_IMAGE, (CS_VOID*)&value);
    if (retval == CS_FAIL)
    {
        return(CS_FAIL);
    }
}

return (CS_SUCCEED);
}

```

#### 使用法

- クライアントから発行可能な要求、および Open Server アプリケーションから返信可能な応答については、Open Server アプリケーションとクライアントがお互いに認識できる必要があります。クライアント/サーバ接続の機能が、その接続について許可されるクライアント要求とサーバ応答の種類を決定します。
- Open Server は、すべての接続について、機能のデフォルト・セットを指定します。特定の接続に対して機能のデフォルト・セットを適用しない場合、Open Server アプリケーションは、`srv_capability_info` を呼び出して異なる機能のセットを明示的にネゴシエートできます。



戻り値

表 3-24 : 戻り値 (srv\_createmsgq)

戻り値 :	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype
*/
CS_RETCODE      ex_srv_createmsgq PROTOTYPE((
    SRV_OBJID    *msgqp,
    CS_CHAR      *msgqnm
));

/*
** EX_SRV_CREATEMSGQ
**
** Example routine to create an Open Server message queue
** using srv_createmsgq.
**
** Arguments:
** msgqp Return pointer to the created message queue
**        identifier.
** msgqnm Null terminated name for the created queue.
**
** Returns:
** CS_SUCCEED Message queue with given name successfully
**             created.
** CS_FAIL    An error was detected.
*/
CS_RETCODE      ex_srv_createmsgq(msgqp, msgqnm)
SRV_OBJID      *msgqp;
CS_CHAR        *msgqnm;
{
    /* Check parameters.*/
    if ((CS_INT)strlen(msgqnm) > SRV_MAXNAME)
    {
        return(CS_FAIL);
    }

    /* Create the message queue.*/
    if (srv_createmsgq(msgqnm, (CS_INT)CS_NULLTERM, msgqp) !=
        CS_SUCCEED)

```

```

    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

使用法

- メッセージ・キューを作成する場合は、メッセージ・キューに名前を付けます。メッセージ・キューが作成されると、名前と ID のどちらでもメッセージ・キューを参照できます。
- メッセージ・キュー ID がわかっている場合は、`srv_getobjname` を使って名前を検索してください。
- `SRV_OBJID` は `CS_INT` として定義されています。
- `SRV_S_NUMMSGQUEUES` サーバ・プロパティは、Open Server アプリケーションが使用できるメッセージ・キューの数を決定します。「サーバ・プロパティ」(159 ページ)を参照してください。
- `SRV_S_MSGPOOL` サーバ・プロパティは、ランタイムに Open Server アプリケーションが使用できるメッセージの数を決定します。「サーバ・プロパティ」(159 ページ)を参照してください。

参照

[srv\\_deletemsgq](#), [srv\\_getmsgq](#), [srv\\_getobjname](#), [srv\\_putmsgq](#)

## srv\_createmutex

説明

相互排他セマフォを作成します。

構文

```

CS_RETCODE srv_createmutex(mutex_namep, mutex_namelen,
                           mutex_idp)

CS_CHAR    *mutex_namep;
CS_INT     mutex_namelen;
SRV_OBJID  *mutex_idp;

```

パラメータ

*mutex\_namep*

作成するミューテックスの名前を指すポインタです。

*mutex\_namelen*

*\*mutex\_namep* 内の名前の長さです。文字列が NULL で終了する場合は、*mutex\_namelen* は `CS_NULLTERM` とすることもできます。

*mutex\_idp*

Open Server は、新しいミューテックスの ID を *\*mutex\_idp* に返します。

戻り値

表 3-25 : 戻り値 (srv\_createmutex)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local Prototype.
**
*/
CS_RETCODE          ex_srv_createmutex PROTOTYPE((
CS_CHAR            *name,
CS_INT             namelen,
SRV_OBJID          *idp
));

/*
** EX_SRV_CREATEMUTEX
**
** Example routine to create an Open Server mutex.
**
** Arguments:
**
** name           The name of the mutex to create.
** namelen        The length of name.
** idp            The address of a SRV_OBJID, which will be set
**                to the unique identifier for the created mutex.
**
** Returns:
** CS_SUCCEED     The mutex was created successfully.
** CS_FAIL        An error was detected.
**
*/
CS_RETCODE          ex_srv_createmutex(name, namelen, idp)
CS_CHAR            *name;
CS_INT             namelen;
SRV_OBJID          *idp;
{
    /*
    ** Call the Open Server routine that will create
    ** the mutex.
    */
    if( srv_createmutex(name, namelen, idp) == CS_FAIL )
    {
        /*
        ** An error was already raised.

```

```

        */
        return CS_FAIL;
    }

    /*
    ** All done.
    */
    return CS_SUCCEEDED;
}

```

- 使用法**
- ミューテックスを作成する場合は、ミューテックスに名前を付けます。ミューテックスが作成されると、名前と ID のどちらでもミューテックスを参照できます。
  - ミューテックスの ID がわかっている場合は、`srv_getobjname` を使って名前を検索します。
  - ミューテックスを作成しても、作成者にロック権が付与されるわけではありません。作成したミューテックスをロックするには、`srv_lockmutex` を使用してください。
  - `SRV_OBJID` は `CS_INT` として定義されています。

**参照** [srv\\_deletemutex](#), [srv\\_getobjname](#), [srv\\_lockmutex](#), [srv\\_unlockmutex](#)

## srv\_createproc

- 説明** 非クライアントのイベント駆動型スレッドを作成します。
- 構文** `SRV_PROC *srv_createproc(ssp)`  
`SRV_SERVER *ssp;`
- パラメータ** `ssp`  
 Open Server ステータス情報の制御構造体へのポインタです。
- 戻り値** 成功した場合は、`srv_createproc` によって新しいスレッド制御構造体にポインタが返されます。失敗した場合は、`srv_createproc` によって NULL スレッド・ポインタが返され、Open Server はエラーとなります。

表 3-26 : 戻り値 (srv\_createproc)

戻り値	意味
新しいスレッド制御構造体へのポインタ	Open Server はスレッドを作成した。
null スレッド・ポインタ	Open Server はスレッドを作成できなかった。 Open Server はエラーになる。

## 例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
CS_RETCODE          ex_srv_creatp          PROTOTYPE((
SRV_SERVER          *ssp,
SRV_PROC            *newsp
));
/*
** EX_SRV_CREATP
**      Example routine to create a non-client, event driven
**      thread.
**
** Arguments:
**
**      ssp          A pointer to the Open Server state information
**                  control structure.
**      newsp        A pointer that will be returned by srv_createproc
**                  and point to the new thread control structure.
**
** Returns
**
**      CS_SUCCEED          Thread was created.
**      CS_FAIL            An error was detected.
**
**/
CS_RETCODE          ex_srv_creatp(ssp, newsp)
SRV_SERVER          *ssp;
SRV_PROC            *newsp;
{
    /* Check arguments.*/
    if(ssp == (SRV_SERVER *)0)
        return(CS_FAIL);

    /*
    ** Create the new thread

```

```

*/
newsp = srv_createproc(ssp);
if(newsp == (SRV_PROC *)NULL)
    return(CS_FAIL);
return (CS_SUCCEED);
}

```

使用法

- `srv_createproc` は、`srv_event` または `srv_event_deferred` から発生するユーザ定義イベントによって駆動される、スレッドを作成します。
- 非クライアント・スレッドは、ユーザ定義イベントのみを受け取り、クライアント生成イベントを受け取りません。
- `srv_createproc` で作成されたスレッドを終了するには、`srv_termproc` を使用してください。
- 非クライアント・スレッドにはクライアント I/O がありません。非クライアント・スレッドの場合は、`property` 引数を (`SRV_T_IODEAD`) に設定して `srv_thread_props` を呼び出すと、常に `CS_FALSE` が返ります。

参照

[srv\\_event](#), [srv\\_event\\_deferred](#), [srv\\_spawn](#), [srv\\_termproc](#), [srv\\_thread\\_props](#)

## srv\_cursor\_props

説明

現在のカーソルに関する情報を取得または設定します。

---

**注意** 15.0 より前のアプリケーションの場合は、`SRV_CURDESC` 構造体および `srv_cursor_props` ルーチンを使用します。バージョン 15.0 およびそれ以降のアプリケーションでは `Open Server` でスクロール可能なカーソルをサポートしているため、`SRV_CURDESC2` 構造体と `srv_cursor_props2` ルーチンを使用します。

---

構文

```

CS_RETCODE srv_cursor_props(spp, cmd, cdp)
SRV_PROC      *spp;
CS_INT        cmd;
SRV_CURDESC   *cdp;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。



*cmd*

`srv_cursor_props` がカーソル情報をクライアントに送信するのか、クライアントからカーソル情報を取得するのかを示します。次の表に、*cmd* の有効値を示します。

表 3-27 : *cmd* の値 (`srv_cursor_props`)

値	説明
CS_SET	<code>srv_cursor_props</code> が現在のカーソルに関する情報をクライアントに送信する。
CS_GET	<code>srv_cursor_props</code> が現在のカーソル・コマンドに関する情報をクライアントから取得する。

*cdp*

`SRV_CURDESC` 構造体へのポインタです。アプリケーションがカーソル情報を設定する場合は、`SRV_CURDESC` 構造体が現在のカーソルを記述します。アプリケーションが情報を取得する場合は、`Open Server` は、現在のカーソルに関する情報を使って `SRV_CURDESC` 構造体を更新します。現在のカーソル・コマンドによって、さまざまなフィールドがそれぞれ異なった時点で設定されたり、データが取り込まれたりします。*cdp* の各フィールドの説明と、それぞれのフィールドにいつどのようにデータが取り込まれるかについては、「[「SRV\\_CURDESC 構造体」\(69 ページ\)](#)」を参照してください。

## 戻り値

表 3-28 : 戻り値 (`srv_cursor_props`)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <ospublic.h>
/*
** Local Prototype.
**/
extern CS_RETCODE ex_srv_cursor_props PROTOTYPE((
CS_VOID *spp
));
/*
** EX_SRV_CURSOR_PROPS
**
** Example routine to retrieve information on the current
** cursor.
** Arguments:
** spp Apointer to an internal control structure.
```

```

**
** Returns:
**
** CS_SUCCEEDED   Cursor information was retrieved successfully.
** CS_FAIL        An error was detected.
*/
CS_RETCODE        ex_srv_cursor_props(spp)
SRV_PROC          *spp;
{
    SRV_CURDESC   curdesc;

    if(srv_cursor_props(spp, CS_GET, &curdesc) == CS_FAIL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

#### 使用法

- Open Server アプリケーションは、クライアントとのアクティブ・カーソル情報の交換に `srv_cursor_props` を使用します。
- 常に、クライアントはカーソル・コマンドを発行することによって、この交換を開始します。つまり、現在のカーソルを指定するのはクライアントです。
- アプリケーションが `srv_cursor_props` を呼び出せるのは **SRV\_CURSOR** イベント・ハンドラ内からのみです。
- Open Server はクライアントから受け取った各カーソル・コマンドでそれぞれ **SRV\_CURSOR** イベントを生成します。これを受けて、アプリケーションの **SRV\_CURSOR** イベント・ハンドラは、*cmd* を **CS\_GET** に設定して `srv_cursor_props` を呼び出すことによって、現在のカーソルと受けたカーソル・コマンドの種類を特定することができます。そうした場合、どのように応答するかを決めることができます。有効なカーソル・コマンドの種類と有効な応答の説明については、「[カーソル](#)」(66 ページ) を参照してください。
- 各カーソル・コマンドは、Open Server アプリケーションから独自の応答を引き出します。アプリケーションは **SRV\_CURDESC** 構造体から情報(要求されたフェッチ・カウントなど)を抜き出し、そのデータに基づいて決定を下し、構造体で情報を設定し、`srv_cursor_props` を使用して情報をクライアントに返します。状況によっては、アプリケーションはパラメータを読み込んだり、結果ローヤパラメータを返したりすることもできます。

- `SRV_CURSOR` イベント・ハンドラは、カーソル情報コマンドを返すことによって、フェッチ、更新、削除を除く、すべてのカーソル・コマンドに応答しなければなりません。ハンドラは `SRV_CURDESC` 構造体の `curcmd` フィールドを `CS_CURSOR_INFO` に設定し、`cmd` を `CS_SET` に設定して `srv_cursor_props` を呼び出します。これが、ハンドラが返す最初の情報です。
- `Open Server` アプリケーションは、`CURSOR_DECLARE` コマンドに回答して、現在のカーソルをユニークに識別できるカーソル ID を選択します。アプリケーションは、`cmd` を `CS_SET` に設定した状態で `srv_cursor_props` を呼び出すことによって、カーソル ID をクライアントに返します。それ以降、クライアントおよび `Open Server` アプリケーションは、現在のカーソルを名前ではなく ID を使って参照します。

参照

[srv\\_bind](#), [srv\\_descfmt](#), [srv\\_numparams](#), [srv\\_xferdata](#), 「カーソル」 (66 ページ)

## srv\_cursor\_props2

説明

現在のカーソルに関する情報を取得または設定します。

---

**注意** バージョン 15.0 およびそれ以降のアプリケーションでは `Open Server` でスクロール可能なカーソルをサポートしているため、`SRV_CURDESC2` 構造体と `srv_cursor_props2` ルーチンを使用します。15.0 より前のアプリケーションの場合は、`SRV_CURDESC` 構造体および `srv_cursor_props` ルーチンを使用します。

---

構文

```
CS_RETCODE srv_cursor_props2(spp, cmd, cdp)
```

```
SRV_PROC      *spp;
CS_INT        cmd;
SRV_CURDESC2 *cdp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

`srv_cursor_props2` がカーソル情報をクライアントに送信するのか、クライアントからカーソル情報を取得するのかを示します。

**表 3-29 : cmd の値 (srv\_cursor\_props2)**

値	説明
CS_SET	srv_cursor_props2 が現在のカーソルに関する情報をクライアントに送信する。
CS_GET	srv_cursor_props2 が現在のカーソル・コマンドに関する情報をクライアントから取得する。

*cdp*

SRV\_CURDESC2 構造体へのポインタです。アプリケーションがカーソル情報を設定する場合は、SRV\_CURDESC2 構造体が現在のカーソルを記述します。アプリケーションが情報を取得する場合は、Open Server は、現在のカーソルに関する情報を使って SRV\_CURDESC2 構造体を更新します。現在のカーソル・コマンドによって、さまざまなフィールドがそれぞれ異なった時点で設定されたり、データが取り込まれたりします。*cdp* の各フィールドの説明と、それぞれのフィールドにいつどのようにデータが取り込まれるかについては、「SRV\_CURDESC2 構造体」(83 ページ)を参照してください。

戻り値

**表 3-30 : 戻り値 (srv\_cursor\_props2)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

使用法

srv\_cursor\_props ルーチンの「使用法」の項を参照してください。

参照

srv\_bind, srv\_descfmt, srv\_numparams, srv\_xferdata, 「カーソル」(66 ページ)

## srv\_dbg\_stack

説明

スレッドのコール・スタックを表示します。

構文

```
CS_RETCODE srv_dbg_stack(spp, depth, funcp)
SRV_PROC   *spp;
CS_INT     depth;
CS_RETCODE (*funcp)();
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*depth*

表示するコール・スタック・レベルの最大数です。*depth* が 1 の場合は、すべてのレベルが表示されます。

*funcp*

コール・スタック表示の各ラインを処理するためにユーザが提供する関数へのポインタです。この関数を呼び出す場合は、`null` で終了する文字列、および文字列の長さである整数を指定します。文字列には、プログラム・カウンタとルーチンのパラメータが 16 進数形式で格納されています。この関数から `CS_FAIL` が返された場合は、スタック・トレースが終了します。`CS_FAIL` 以外が返された場合は、コール・スタックのすべてのルーチンが処理されるか、*depth* スタック・フレームが処理されるまで、スタック・トレースが続きます。*funcp* が `NULL` の場合は、`Open Server` はコール・スタックの内容を `stderr` に書き込みます。

次に、一般的な例を示します。

```
CS_RETCODE callstack_display(linebuf, length)
CS_CHAR *linebuf;
CS_INT length;
{
    /*
    ** Output each line of the stack trace to stderr.
    */
    fprintf(stderr, "%s¥n", linebuf);
    return (CS_SUCCEEDED);
}
```

戻り値

表 3-31 : 戻り値 (`srv_dbg_stack`)

戻り値	意味
<code>CS_SUCCEEDED</code>	ルーチンが正常に終了した。
<code>CS_FAIL</code>	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local prototype.
*/
CS_RETCODE ex_srv_dbg_stack PROTOTYPE((
SRV_PROC *spp
));

/*
** EX_SRV_DBG_STACK
```

```

**
**      Example routine to display the call stack of a thread.
**
** Arguments:
**      spp      - A pointer to an internal thread control structure.
**
** Returns:
**      CS_SUCCEEDED   Call stack successfully displayed.
**      CS_FAIL        An error was detected.
**
*/
CS_RETCODE      ex_srv_dbg_stack(spp)
SRV_PROC        *spp;
{
    CS_RETCODE retval;

    retval = srv_dbg_stack(spp, -1, (CS_RETCODE(*)())NULL);

    return (retval);
}

```

#### 使用法

- `srv_dbg_stack` は、すべてのプラットフォームで使用できるわけではありません。`srv_capability` を使って、現在のプラットフォームで使用可能かどうかを調べてください。
- `srv_dbg_stack` を使用すると、デバッグ中、または実行エラーの処理中に、スレッドのコール・スタックを検査することができます。`srv_dbg_stack` は、デバッガ、または実行中のアプリケーションから呼び出すことができます。
- 通常、`srv_dbg_stack` は、重大なエラーが生じたときにスタック・フレームをエラー・ログに記録するために使用します。
- 呼び出しスタックの各ルーチンは、16 進数のプログラム・カウンタ、続いて同じく 16 進数の各パラメータで構成される文字列にフォーマットされます。プログラム・カウンタを関数名に変換するには、実行プログラムのロード・マップが必要です。
- 現在実行しているスレッドのスタックを表示するように呼び出すと、`srv_dbg_stack` およびそのコール・ルーチンはスタック上に表示されます。

#### 参照

[srv\\_capability](#), [srv\\_dbg\\_switch](#)

## srv\_dbg\_switch

**説明** デバッグのために、もう 1 つのスレッド・コンテキストを一時的にリストアします。

**構文** CS\_RETCODE srv\_dbg\_switch(spid)

CS\_INT spid;

**パラメータ**

*spid*

コンテキストが一時的にリストアするスレッドのサーバ・プロセス ID (*spid*) です。

**戻り値**

**表 3-32 : 戻り値 (srv\_dbg\_switch)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**使用法**

- `srv_dbg_switch` は、すべてのプラットフォームで使用できるわけではありません。`srv_capability` を使って、プラットフォームで `srv_dbg_switch` をサポートしているかどうかを調べてください。
- スレッド・コンテキストが切り替えられると、アプリケーションを実行し続けた場合、元のスレッド・コンテキストがリストアされ、アプリケーションは正常に実行し続けます。
- コンテキストがリストアされているスレッドは、実行できません。この場合、スレッドは調べる対象にしかありません。
- UNIX システムの場合、システム・サービス・ルーチン内から `srv_dbg_switch` を呼び出さないでください。呼び出した場合は、SIGTRAP シグナルが発行され、プログラムは終了します。
- *spid* を取得する場合は、`property` 引数を `SRV_T_SPID` に設定して `srv_thread_props` を呼び出します。現在実行しているスレッドのコンテキストをリストアしようとする、エラーになります。

**参照**

[srv\\_capability](#), [srv\\_dbg\\_stack](#)

## srv\_define\_event

**説明** ユーザ・イベントを定義します。

**構文** int srv\_define\_event(ssp, type, namep, namelen)

```
SRV_SERVER    *ssp;
CS_INT        type;
CS_CHAR       *namep;
CS_INT        namelen;
```

パラメータ

*ssp*

Open Server の制御構造体へのポインタです。

*type*

イベントのタイプです。現在、ユーザ定義イベントは SRV\_QUEUED のタイプでなければなりません。

*namep*

イベント名へのポインタです。

*namelen*

\*namep の文字列の長さをバイト数で示したものです。文字列が null で終了する場合、namelen は CS\_NULLTERM とすることもできます。

戻り値

**表 3-33 : 戻り値 (srv\_define\_event)**

戻り値	意味
0 でない整数	イベント用のユニークな <b>id</b> 。
0	Open Server はイベントを定義できない。 Open Server はエラーになる。

例

```
#include          <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE       ex_srv_define_event PROTOTYPE((
CS_CHAR          *namep,
CS_INT           namelen,
CS_INT           *event_no
));
/*
** EX_SRV_DEFINE_EVENT
**
** Example routine to illustrate the use of srv_define_event to
** define an user event.
**
** Arguments:
** namep   A pointer to the name of event.
** namelen The length, in bytes, of string in *namep.
```



```

**      event_no  A CS_INT pointer that is initialized with
**                the unique number for the event.
** Returns:
**
**  CS_SUCCEED   If the event was defined successfully.
**  CS_FAIL      An error was detected.
*/
CS_RETCODE      ex_srv_define_event(namep, namelen, event_no)
CS_CONTEXT      *cp;
CS_VOID         *bufp;
CS_CHAR         *namep;
CS_INT          namelen;
CS_INT          *event_no;
CS_INT          result;
{
    SRV_PROC *srvproc_ptr; /* A pointer to an internal thread
                           ** control structure */

    result = srv_props(cp, CS_GET, SRV_S_CURTHREAD,
                      bufp, sizeof(CS_INT));
    if (result == CS_FAIL)
    {
        return(CS_FAIL);
    }
    /* Now define the event.*/
    if ((*event_no = srv_define_event(srvproc_ptr, SRV_QUEUED,
                                     namep, namelen)) == (CS_INT)0)
        return(CS_FAIL);
    return (CS_SUCCEED);
}

```

#### 使用法

- ユーザ定義イベントは、クライアント・アクションによってトリガされるのではなく、`srv_event` を呼び出すことによってトリガされます。**Open Server** プログラムは、イベントがトリガされたときに実行するハンドラ・ルーチンを提供します。
- ユーザ定義イベントのイベント・ハンドラは、通常、`srv_handle` を使ってインストールします。
- ユーザ定義イベントのハンドラは、イベントを受けたスレッドのスレッド制御構造体へのポインタを受け取ります。
- ユーザ定義イベントが許可されるように **Open Server** アプリケーションを設定していない場合は、イベントを定義できません。詳細については、`srv_props` のページを参照してください。

#### 参照

`srv_event`, `srv_event_deferred`, `srv_handle`, `srv_props`, 「イベント」 (100 ページ)

## srv\_deletemsgq

**説明**                                   メッセージ・キューを削除します。

**構文**                                   CS\_RETCODE srv\_deletemsgq(msgqnamep, msgqname\_len,  
  msgqid)  
  
CS\_CHAR        \*msgqnamep;  
CS\_INT         msgqname\_len;  
SRV\_OBJID     msgqid;

**パラメータ**                           *msgqnamep*  
削除するメッセージ・キュー名へのポインタです。存在しないメッセージ・キューを削除しようとする、エラーになります。

*msgqname\_len*  
*msgqname* によって示される、名前の長さです。名前が NULL で終了する場合、*msgqname\_len* は CS\_NULLTERM とすることもできます。

*msgqid*  
削除するメッセージ・キューの識別子を指定する SRV\_OBJID です。

**戻り値**                                   **表 3-34 : 戻り値 (srv\_deletemsgq)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

### 例

```
#include <ospublic.h>
/*
** Local Prototype.
**/
CS_RETCODE ex_srv_deletemsgq PROTOTYPE((
CS_CHAR *msgqname,
CS_INT msgqname_len,
SRV_OBJID msgqid
));
/*
** EX_SRV_DELETEMSGQ
**
** Example routine using srv_deletemsgq to delete an Open
** Server message queue previously create by srv_createmsgq.
** This routine can be passed a value message queue name, or
** NULL, in which case the message queue identifier will be used.
**
** Arguments:
```

```

** msgqname          The name of the message queue to delete.If
**                   NULL, the msgqid is used.
** msgqname_len      The length of the name to which msgqname
**                   points.
** msgqid            A SRV_OBJID that specifies the identifier of
**                   the message queue to delete.
**
** Returns:
**
** CS_SUCCEEDED      The message queue was successfully deleted.
** CS_FAIL           An error was detected.
*/
CS_RETCODE           ex_srv_deletemsgq(msgqname, msgqname_len, msgqid)
CS_CHAR              *msgqname;
CS_INT               msgqname_len;
SRV_OBJID            msgqid;
{
    /*
     ** Delete a message queue.
     */
    if (srv_deletemsgq(msgqname, msgqname_len, msgqid) !=
        CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

## 使用法

- メッセージ・キューを削除する場合は、名前または ID を使用します。*msgqname* が NULL でない場合は、メッセージ・キュー名を使用します。そうでない場合は、メッセージ・キュー ID を使用します。
- キュー内の未読メッセージは、キューが削除される前にフラッシュされます。*srv\_putmsgq* で待機しているスレッドがウェイクアップし、*srv\_putmsgq* は CS\_FAIL を返します。
- メッセージ・キューが削除されると、そのキューからのメッセージを待っているスレッドは *srv\_getmsgq* からの CS\_FAIL 戻り値でウェイクアップし、*srv\_getmsgq* の *infop* 引数が SRV\_I\_DELETED に設定されます。
- SRV\_S\_NUMMSGQUEUES サーバ・プロパティは、Open Server アプリケーションが使用できるメッセージ・キューの数を決定します。「サーバ・プロパティ」(159 ページ) を参照してください。

- SRV\_S\_MSGPOOL サーバ・プロパティは、ランタイムに Open Server アプリケーションが使用できるメッセージの数を決定します。「サーバ・プロパティ」(159 ページ) を参照してください。

参照

[srv\\_createmsgq](#), [srv\\_getmsgq](#), [srv\\_getobjname](#), [srv\\_putmsgq](#)

## srv\_deletemutex

**説明** `srv_createmutex` によって作成された ミューテックスを削除します。

**構文**

```
CS_RETCODE srv_deletemutex(mutex_namep, mutex_namelen,
                             mutex_id)
CS_CHAR     *mutex_namep;
CS_INT      mutex_namelen;
SRV_OBJID   mutex_id;
```

**パラメータ**

*mutex\_namep*

ミューテックス作成時にミューテックスに関連付けられた、名前へのポインタです。

*mutex\_namelen*

*mutex\_namep* の長さをバイト数で示したものです。文字列が NULL で終了する場合、*mutex\_namelen* は CS\_NULLTERM とすることもできます。

*mutex\_id*

`srv_createmutex` が返すユニークな識別子です。

**戻り値**

**表 3-35 : 戻り値 (srv\_deletemutex)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include <ospublic.h>
/*
** Local Prototype.
*/
CS_RETCODE ex_srv_deletemutex PROTOTYPE((
CS_CHAR     *mtxnm,
SRV_OBJID   mtxid
));
```

```

/*
** EX_SRV_DELETEMUTEX
** Example routine using srv_deletemutex to delete an
** Open Server mutex previously created by srv_createmutex.
** This routine can be passed a mutex name, or NULL,
** in which case the mutex identifier will be used.
** Arguments:
** mtxnm      Null terminated mutex name, or NULL to use mutex
**            id.
** mtxid      Mutex identifier (valid only if mtxnm is NULL).
** Returns:
** CS_SUCCEED  mutex was successfully queued for deletion.
** CS_FAIL     An error was detected.
*/
CS_RETCODE      ex_srv_deletemutex(mtxnm, mtxid)
CS_CHAR         *mtxnm;
SRV_OBJID       mtxid;
{
    /* Delete the mutex.*/
    if (srv_deletemutex(mtxnm, (CS_INT)CS_NULLTERM, mtxid) !=
        CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

**使用法**

- ミューテックスを削除する場合は、名前または ID を使用します。*mutex\_namep* が NULL でない場合は名前を、そうでない場合は ID を使用します。
- 他のスレッドがミューテックスをロックしようとして待機している場合は、そのスレッドの要求が満たされてロックが解放されるまでは、ミューテックスは削除されません。
- ミューテックスの使用例が、*srv\_createmutex* のページに示されています。

**参照**

[srv\\_createmutex](#), [srv\\_getobjid](#), [srv\\_getobjname](#), [srv\\_lockmutex](#)

## srv\_descfmt

**説明**

クライアントとやり取りするカラムまたはパラメータの記述を、記述または取得します。

構文

```

CS_RETCODE srv_descfmt(spp, cmd, type, item,
                        clfmt)
SRV_PROC      *spp;
CS_INT        cmd;
CS_INT        type;
CS_INT        item;
CS_DATAFMT    *clfmt;
    
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

*srv\_descfmt* はクライアントに送られるデータを記述するのか、クライアントから受けるデータの記述を取得するのかを示します。

表 3-36 に、*cmd* の有効値を示します。

**表 3-36 : *cmd* の値 (*srv\_descfmt*)**

値	説明
CS_SET	<i>srv_descfmt</i> は、クライアントがデータを受け取る時のフォーマットを記述する。
CS_GET	<i>srv_descfmt</i> は、クライアントがデータを送った時のフォーマットを取得する。

*type*

*cmd* が CS\_SET の場合は、記述されているデータの型です。*cmd* が CS\_GET の場合は、取得されているデータの型です。表 3-37 に、有効な型とその適切なコンテキストを示します。

表 3-37 : type の値 (srv\_descfmt)

Type	cmd の許容される設定	説明
SRV_RPCDATA	CS_SET または CS_GET	RPC またはストアド・プロシージャ・パラメータ
SRV_ROWDATA	CS_SET のみ	ローのデータ
SRV_CURDATA	CS_GET のみ	カーソル・パラメータ
SRV_UPCOLDATA	CS_GET のみ	カーソル更新カラム
SRV_KEYDATA	CS_GET のみ	カーソル・キー・データ
SRV_ERRORDATA	CS_SET のみ	拡張エラー・データ
SRV_DYNDATA	CS_SET または CS_GET	動的 SQL データ
SRV_NEGDATA	CS_SET または CS_GET	ネゴシエーション・ログイン・データ
SRV_MSGDATA	CS_SET または CS_GET	MSG パラメータ
SRV_LANGDATA	CS_GET のみ	言語パラメータ

*item*

パラメータまたはカラムの番号です。パラメータやカラムの番号は 1 から始まります。

*clfmt*

データの記述が格納されている CS\_DATAFMT 構造体へのポインタです。

## 戻り値

表 3-38 : 戻り値 (srv\_descfmt)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE ex_srv_descfmt PROTOTYPE((
SRV_PROC *spp,
CS_INT item,
CS_DATAFMT *dp
));

/*
```

```

** EX_SRV_DESCFMT
**
**      Example routine used to get an RPC parameter description.
**
** Arguments:
**
**      spp          A pointer to an internal thread control
**                  structure.
**      item        The parameter number we're looking for.
**      dp          The address of a CS_DATAFMT to be filled with
**                  the parameter's description.
**
** Returns:
**      CS_SUCCEED if the description was obtained, or
**      CS_FAIL if an error was detected.
*/
CS_RETCODE      ex_srv_descfmt(sp, item, dp)
SRV_PROC        *sp;
CS_INT          item;
CS_DATAFMT      *dp;
{
    /*
    ** Call srv_descfmt to get the RPC parameter description.
    */
    if( srv_descfmt(sp, CS_GET, SRV_RPCDATA, item, dp) ==
        CS_FAIL )
    {
        /*
        ** An error was already raised.
        */
        return CS_FAIL;
    }

    /*
    ** All done.
    */
    return CS_SUCCEED;
}

```

#### 使用方法

- `srv_descfmt` は、さまざまなカラムやパラメータのフォーマットを記述します。詳細については、「[CS\\_DATAFMT 構造体](#)」(57 ページ)を参照してください。



- クライアントにローヤパラメータを送信する場合 (CS\_SET)、クライアントへのデータの表示方法を記述するために `srv_descfmt` を呼び出します。クライアントからパラメータを受け取る場合 (CS\_GET)、クライアントがデータを送信したときのフォーマットの記述を取得するために `srv_descfmt` を呼び出します。このクライアントのフォーマット情報については、リモート・サーバに渡せるように、ゲートウェイ・アプリケーションが保存する場合があります。
- `srv_descfmt` ルーチンは、次の表に示す CS\_DATAFMT フィールドに対して、読み取り (CS\_GET) や設定 (CS\_SET) を行います。他のすべてのフィールドは、`srv_descfmt` には未定義です (“`clfmtmp`” は構造体を指すポインタであることに注意してください)。

表 3-39 : 使用される CS\_DATAFMT フィールド (`srv_descfmt`)

フィールド	CS_SET	CS_GET
<code>clfmtmp</code> → <code>namelen</code>	名前の長さ	名前の長さ
<code>clfmtmp</code> → <code>status</code>	パラメータ/カラム・ステータス	パラメータ・ステータス
<code>clfmtmp</code> → <code>name</code>	パラメータ/カラム名	パラメータ名
<code>clfmtmp</code> → <code>datatype</code>	リモート・データ型をここに設定	リモート・データ型をここから取得
<code>clfmtmp</code> → <code>maxlength</code>	リモート・データ型の最大長をここに設定	リモート・データ型の最大長をここから取得
<code>clfmtmp</code> → <code>format</code>	リモート・データ型のフォーマット	リモート・データ型のフォーマット

- CS\_DATAFMT 構造体で記述されたフォーマット (`clfmtmp`) が、その後続く `srv_bind` の呼び出しで記述されるフォーマット (`osfmtmp`) と異なる場合、Open Server は、`cmd` が CS\_SET のときにはクライアント・フォーマット (`clfmtmp`) に、`cmd` が CS\_GET のときにはアプリケーション・フォーマット (`osfmtmp`) に、自動的に変換されます。
- データ・ストリームの各カラムやパラメータが一度記述されバインドされたら、プログラム変数のデータをクライアントに送ったり、クライアントからのデータでプログラム変数を更新したりするために、`srv_xferdata` を呼び出してください。
- SRV\_NEGDATA パラメータは、`srv_negotiate` が返ったあとで、ネゴシエーション・ログイン・オペレーションの一環として送信または受信できるようになります。
- キー・カラム番号は、ローのキー・カラム番号に対応します。

参照 [srv\\_bind](#), [srv\\_cursor\\_props](#), [srv\\_dynamic](#), [srv\\_msg](#), [srv\\_negotiate](#), [srv\\_numparams](#), [srv\\_sendinfo](#), [srv\\_xferdata](#), 「CS\_DATAFMT 構造体」(57 ページ)

## srv\_dynamic

説明 クライアントの動的 SQL コマンドを読み込む、またはコマンドに回答します。

構文 CS\_RETCODE `srv_dynamic(spp, cmd, item, bufp, buflen, outlenp)`

```
SRV_PROC    *spp;
CS_INT      cmd;
CS_INT      item;
CS_VOID     *bufp;
CS_INT      buflen;
CS_INT      *outlenp
```

パラメータ *spp* 内部スレッド制御構造体へのポインタです。

*cmd* 動的コマンドがクライアントから読み出されているのか、クライアントに送られているのかを示します。表 3-40 に、*cmd* の有効値を示します。

**表 3-40 : cmd の値 (srv\_dynamic)**

値	説明
CS_SET	<code>srv_dynamic</code> は、動的コマンドの応答をクライアントに返す。
CS_GET	<code>srv_dynamic</code> は、動的コマンドをクライアントから読み出す。

*item* どのような情報が送信または取得されているのかを示します。表 3-41 に、*item* の有効値を示します。

表 3-41 : tem の値 (srv\_dynamic)

値	意味
SRV_DYN_TYPE	実行されている動的オペレーションの種類
SRV_DYN_IDLEN	動的文の ID の長さ
SRV_DYN_ID	動的文の ID
SRV_DYN_STMTLEN	動的文の長さ
SRV_DYN_STMT	準備または実行されている動的文

*bufp*

*item* 値が返される (CS\_GET) か、設定される (CS\_SET) バッファへのポインタです。

*buflen*

\**bufp* バッファのバイト単位の長さです。表 3-42 に、必要なバッファ・サイズを示します。

表 3-42 : 必要なバッファ・サイズ (srv\_dynamic)

値	要求されるフォーマット (サイズ)
SRV_DYN_TYPE	(CS_INT) のサイズ
SRV_DYN_IDLEN	(CS_INT) のサイズ
SRV_DYN_ID	可変。まず <i>item</i> を CS_DYN_IDLEN に設定して <i>srv_dynamic</i> を呼び出して長さを確定し、それに合ったバッファ・サイズを割り付ける。
SRV_DYN_STMTLEN	(CS_INT) のサイズ
SRV_DYN_STMT	可変。まず <i>item</i> を CS_DYN_STMTLEN に設定して <i>srv_dynamic</i> を呼び出して長さを確定し、それに合ったバッファ・サイズを割り付ける。

*outlenp*

クライアントからデータを取得しているときに (*cmd* は CS\_GET) \**bufp* にコピーされたデータの実際の長さ設定された整数変数へのポインタです。*cmd* が CS\_SET の場合には、この引数は必要ではありません。

戻り値

表 3-43 : 戻り値 (srv\_dynamic)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
```

```

** Local Prototype
*/
extern CS_RETCODE          ex_srv_dynamic PROTOTYPE((
CS_VOID          *spp,
CS_INT           *optypep
));
/*
** EX_SRV_DYNAMIC
**
** Example routine to retrieve dynamic operation type from a
** client.
**
** Arguments:
** spp          Thread control structure.
** optypep     Dynamic operation type.
**
** Returns:
**
** CS_SUCCEED  Dynamic information was retrieved
**              successfully.
** CS_FAIL     An error was detected.
*/
CS_RETCODE      ex_srv_dynamic(spp, optypep)
SRV_PROC        *spp;
CS_INT          *optypep;
{
CS_INT          outlen;

    if(srv_dynamic(spp, CS_GET, SRV_DYN_TYPE, optypep,
        sizeof(*optypep), &outlen) == CS_FAIL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

**使用方法**

- srv\_dynamic ルーチンでは、Open Server アプリケーションが動的 SQL コマンドを読み込んだり、応答を送ることができます。
- 有効なオペレーションの種類 (SRV\_DYN\_TYPE) は、次に示すとおりです。

CS\_PREPARE — 文を準備します (CS\_GET のみ)。

CS\_DESCRIBE\_INPUT — 現在の準備文に使用する入力パラメータ・フォーマットを要求します (CS\_GET のみ)。

CS\_DESCRIBE\_OUTPUT – 現在の準備文に使用するカラム・フォーマットを要求します (CS\_GET のみ)。

CS\_EXECUTE – 準備文を実行します (CS\_GET のみ)。

CS\_EXEC\_IMMEDIATE – パラメータがなく結果を返さない、準備されていない文を実行します (CS\_GET のみ)。

CS\_DEALLOC – 準備文の割り付けを解除します (CS\_GET のみ)。

CS\_ACK – クライアントからの動的 SQL コマンドに応答します (CS\_SET のみ)。

- クライアントから受け取る動的コマンドはすべて、SRV\_DYNAMIC イベントをトリガします。その時点で、Open Server アプリケーションは、各クライアント動的コマンドにตอบสนองしてオペレーション・タイプ、文の ID および文を取得し保存するために `srv_dynamic` を呼び出すことが可能になり、`type` を CS\_ACK に設定して `srv_dynamic` 呼び出しを発行することによって、クライアント通信にตอบสนองすることができます。
- SRV\_DYNAMIC ハンドラ以外の、いかなるハンドラにおいても `srv_dynamic` を呼び出すとエラーが発生します。
- CS\_ACK が、唯一設定可能な動的オペレーション・タイプです (`cmd` を CS\_SET に設定)。
- CS\_PREPARE、CS\_DESCRIBE\_INPUT、CS\_DESCRIBE\_OUTPUT、CS\_EXECUTE、CS\_EXEC\_IMMEDIATE、そして CS\_DEALLOC のみが、取得可能な動的オペレーション・タイプです (`cmd` を CS\_GET に設定)。
- 完全な動的 SQL 応答をクライアントに送ることは、ID 長、ID、オペレーション・タイプを渡すことによって行われます。このためには、`srv_dynamic` に対する3つの個別の呼び出しが必要です。たとえば、文の ID のみを設定して `srv_senddone` を呼び出すと、エラーとなります。唯一の例外が、オペレーション・タイプが CS\_EXEC\_IMMEDIATE の場合で、これには関連する文の ID がありません。
- パラメータ・データ・フォーマットおよび出力フォーマットは、CS\_PREPARE 動的コマンドにตอบสนองして、`type` 引数を SRV\_DYNDATA にして `srv_descfmt` と `srv_xferdata` を使用して、クライアントに送信することができます。これはアプリケーションが単にフォーマットを送信しているだけなので、`srv_bind` は必要ではないことに注意してください。

- Open Server アプリケーションは、クライアントが CS\_EXECUTE 動的コマンドに続いて送信したパラメータ・データを、type 引数を SRV\_DYNDATA にして srv\_descfmt、srv\_bind、srv\_xferdata を使用して、取得し保存します。パラメータの数は、アプリケーションが srv\_numparams を使用して決めます。
- アプリケーションは、type 引数を SRV\_ROWDATA にした状態で、srv\_descfmt、srv\_bind、および srv\_xferdata を使用して CS\_EXECUTE 動的 SQL コマンドに回答し、動的 SQL 結果ローをクライアントに送信します。
- CS\_EXEC\_IMMEDIATE の動的 SQL コマンドは、クライアントがパラメータなしで文を実行し、結果として DONE のみを受け取ることを希望していることを示しています。この文は CS\_EXEC\_IMMEDIATE コマンド・ストリームに含まれ、SRV\_DYN\_STMT を介してアクセスすることが可能です。文は前もって準備されておらず(文の ID 長 (SRV\_DYN\_IDLEN) は 0 になる)、SRV\_DYNAMIC イベント・ハンドラが終了すると、存在しなくなります。

参照 [srv\\_bind, srv\\_descfmt, srv\\_numparams, srv\\_xferdata, 「動的 SQL」\(90 ページ\)](#)

## srv\_envchange

説明 環境の変化をクライアントに通知します。

構文 CS\_RETCODE srv\_envchange(spp, type, oldvalp, oldvallen, newvalp, newvallen)

```
SRV_PROC *spp;
CS_INT type;
CS_CHAR *oldvalp;
CS_INT oldvallen
CS_CHAR *newvalp;
CS_INT newvallen
```

パラメータ *spp* 内部スレッド制御構造体へのポインタです。

*type* 変更されている環境です。現在、有効値は SRV\_ENVDATABASE と SRV\_ENVLANG のみで、それぞれ、現在のデータベース名と各国言語を表します。

*oldvalp*

古い値が格納されている文字列へのポインタです。NULL の場合もあります。ポインタの長さは、バイト数で *oldvallen* に保存されます。

*oldvallen*

\**oldvalp* の文字列の長さをバイト数で示したものです。長さが CS\_NULLTERM の場合は、\**oldvalp* の文字列は NULL で終了します。また、CS\_UNUSED の場合は、\**oldvalp* の文字列が NULL であることを示しています。

*newvalp*

環境変数の新しい値が格納されている文字列へのポインタです。NULL の場合もあります。ポインタの長さは、バイト数で *newvallen* に保存されます。

*newvallen*

\**newvalp* の文字列の長さをバイト数で示したものです。長さが CS\_NULLTERM の場合は、\**newvalp* の文字列は NULL で終了します。また、CS\_UNUSED の場合は、\**newvalp* の文字列が NULL であることを示しています。

## 戻り値

表 3-44 : 戻り値 (srv\_envchange)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include      <ospublic.h>
/*
** Local Prototype.
**
CS_RETCODE  ex_srv_envchange PROTOTYPE((
SRV_PROC    *spp
));
/*
** EX_SRV_ENVCHANGE
**
**      Example routine to notify the client of an environment
**      change.
**
** Arguments:
**      spp      A pointer to an internal thread control structure.
**
** Returns:
```

```

**      CS_SUCCEEDED      Successfully notified client of environment
**                          change.
**      CS_FAIL           An error was detected.
**
*/
CS_RETCODE   ex_srv_envchange(spp)
SRV_PROC     *spp;
{
    CS_RETCODE retval;
    /*
    ** Notify the client that we've changed the database
    ** from "master" to "pubs2".
    */
    retval = srv_envchange(spp, SRV_ENVDATABASE, "master",
        CS_NULLTERM, "pubs2", CS_NULLTERM);
    return (retval);
}

```

使用方法

- さまざまな環境変数を設定できます。Open Server では一部の環境変数が自動的に処理されますが、自動処理されない環境変数については Open Server アプリケーションが処理する必要があります。現行バージョンのアプリケーションにできることは、現在のデータベースや各国言語に関する変更をクライアントに通知することだけです。
- これらの値が変わるたびに、Open Server は Open Server アプリケーションのエラー・ハンドラを呼び出します。Open Server アプリケーションが `srv_envchange` によってこれを変更することも、Open Server が内部コードを使用して変更することもできます。また、その両方も可能です。エラー・ハンドラに渡されるエラー番号は、これらの値のどれかが変わるときにクライアントに返送される Adaptive Server Enterprise メッセージ番号です。これにより、クライアント・アプリケーションは、クライアントが Open Server または Adaptive Server Enterprise のいずれに接続しているかに関係なく、同じメッセージ番号を変更値に一致させることができます。表 3-45 に、メッセージ番号と、それぞれの変更値に相当する `oserror.h` #define を示します。



表 3-45 : 環境変数 (srv\_envchange)

変更される値	メッセージ番号	oserror.h の #define
現在のデータベース	5701	SQLSRV_ENVDB
各国言語	5703	SQLSRV_ENVLANG

## srv\_event

**説明** スレッドの要求処理キューに、イベント要求を追加します。

**構文** CS\_INT srv\_event(spp, event, datap)

```
SRV_PROC *spp;
CS_INT event;
CS_VOID *datap;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*event*

クライアントのイベント・キューに追加するイベント用のトークンです。定義されたイベントのリストについては、「[イベント](#)」(100 ページ)を参照してください。

*datap*

Open Server プログラマによって提供されたデータへのポインタ (CS\_VOID) です。アプリケーションは、イベント・ハンドラ内から、property を SRV\_T\_EVENTDATA に設定して srv\_thread\_props を呼び出すことによって、データを取得することができます。

**戻り値**

表 3-46 : 戻り値 (srv\_event)

戻り値	意味
要求されたイベントのためのトークン	Open Server が新しいイベントを追加した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE ex_srv_event PROTOTYPE((
SRV_PROC *spp,
CS_INT event,
```

```

CS_VOID      *datap
));
/*
** EX_SRV_EVENT
**
** Example routine to queue an event request to an Open Server
** thread's request-handling queue.
**
** Note that if the event is an user-defined one, it
** must have been defined earlier using srv_define_event.
**
** Arguments:
** spp          A pointer to a control structure for an Open
**              Server thread.
** event        The token for the event to be added to the queue.
** datap        Data pointer.
**
** Returns:
**
** CS_SUCCEED   The event was queued successfully
** CS_FAIL      An error was detected.
*/
CS_RETCODE  ex_srv_event(spp, event, datap)
SRV_PROC    *spp;
CS_INT      event;
CS_VOID     *datap;
{
    if (srv_event(spp, event, datap) == CS_FAIL)
        return(CS_FAIL);
    else
        return (CS_SUCCEED);
}

```

#### 使用法

- 特定のクライアント・スレッドのイベント・キューに、イベント要求を追加します。通常、イベント要求は、クライアント・アプリケーションからの **Client-Library** 呼び出しなどで自動的にイベント要求キューに追加されます。ただし、**Open Server** プログラムは、**srv\_event** を使用して具体的に要求を追加できます。

**srv\_event** を使用して、次のイベントをイベント・キューに追加できます。

- **SRV\_DISCONNECT**
- **SRV\_URGDISCONNECT**
- **SRV\_STOP**

- ユーザ定義のイベント
- `srv_handle` は、イベント発生時にどのイベント・ハンドラを呼び出すかを Open Server に通知します。特定のイベントについてハンドラが定義されていない場合は、デフォルトの Open Server イベント・ハンドラが呼び出されます。
- `SRV_URGDISCONNECT` イベントは、Open Server アプリケーションの `SRV_DISCONNECT` イベント・ハンドラを呼び出します。
- `SRV_URGDISCONNECT` イベントは、緊急イベントとしてキューイングされます。そのため、切断イベントは、すでにキューイングされているイベントよりも優先され、スレッドのイベント・キューの先頭に配置されます。これは、Open Server スレッドを即刻中止する場合に役立ちます。
- ユーザ定義のイベントの場合は、トリガ可能となる前に、まず `srv_define_event` で定義する必要があります。
- `srv_event` は、`SRV_STOP` または `SRV_START` 以外のイベントをスレッドのイベント・キューに追加します。`SRV_STOP` または `SRV_START` イベントの場合、`spp` は、イベントを要求しているスレッドの内部スレッド制御構造体を指します。
- Open Server アプリケーションは、I/O として機能するルーチンをユーザ定義イベント内からは呼び出せません。

---

**警告！** 割り込みレベルのコードでは、`srv_event` ではなく `srv_event_deferred` を使用します。

---

## 参照

[srv\\_define\\_event](#), [srv\\_handle](#), [srv\\_event\\_deferred](#), [srv\\_thread\\_props](#), 「イベント」(100 ページ)

## srv\_event\_deferred

説明	非同期イベントの結果として、スレッドのイベント・キューにイベント要求を追加します。
構文	<pre>CS_INT srv_event_deferred(spp, event, datap) SRV_PROC *spp; CS_INT event; CS_VOID *datap;</pre>

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*event*

スレッドのイベント・キューに追加するイベントです。

*datap*

Open Server プログラマによって提供されたデータへのポインタ (CS\_VOID) です。アプリケーションは、イベント・ハンドラ内から、*property* を SRV\_T\_EVENTDATA に設定して *srv\_thread\_props* を呼び出すことによって、データを取得することができます。

戻り値

要求されたイベントを返します。エラーの場合は、-1 が返されます。

**表 3-47 : 戻り値 (srv\_event\_deferred)**

戻り値	意味
要求されたイベントのためのトークン	Open Server が新しいイベントを追加した。
-1	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
CS_RETCODE      ex_srv_event_deferred PROTOTYPE((
SRV_PROC        *spp,
CS_INT          event,
CS_VOID         *datap
));
/*
** EX_SRV_EVENT_DEFERRED
** Example routine to queue up a deferred event using
** srv_event_deferred. A deferred event request will
** typically be made from within interrupt-level code.
** Arguments:
** spp          A pointer to the internal thread control
**               structure.
** event        The event to add to the thread's queue.
** datap        A pointer to data to attach to the event.
** Returns:
** CS_SUCCEED   The event was successfully queued.
** CS_FAIL      An error was detected.
**/
CS_RETCODE      ex_srv_event_deferred(spp, event, datap)
SRV_PROC        *spp;
```

```

CS_INT      event;
CS_VOID     *datap;
{
    /*
    ** Add a deferred event to the event queue.
    */
    if (srv_event_deferred(spp, event, datap) == -1)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

#### 使用法

- `srv_event_deferred` は、たとえば UNIX におけるシグナルの配信などのように、割り込みレベル・コードからスレッドのイベント・キューにイベント要求を追加します。イベント要求は、`srv_event_deferred` が呼び出されたときに、Open Server の内部的な重要な機能が実行中の場合は、それらがすべて終了するまで遅延されます。
- Open Server アプリケーションによっては、割り込みレベル・コードからイベントを発生させることができなければなりません。たとえば、アテンション・ハンドラ内でイベントを発生させたい場合や、Open Server アプリケーション・コードでアラーム・シグナルを使用している場合は、`srv_event` の代わりに `srv_event_deferred` を使用する必要があります。`srv_event_deferred` は、リンクされたリストの更新や内部的なハウスキーピング実行などの重要な機能が、イベント要求が実行される前に完了することを保証します。

---

**警告！** 割り込みレベルのコードでは、`srv_event` ではなく `srv_event_deferred` を使用します。

---

- Open Server は通常、スレッドのイベント要求キューに自動的にイベント要求を追加します。ただし、`srv_event_deferred` で具体的な要求を追加することもできます。
- 次のイベントは、`srv_event_deferred` でイベント・キューに追加できます。
  - `SRV_DISCONNECT`
  - `SRV_URGDISCONNECT`
  - `SRV_STOP`
  - ユーザ定義のイベント

- `srv_handle` は、イベント発生時にどのイベント・ハンドラを呼び出すかを **Open Server** に通知します。特定のイベントについてハンドラが定義されていない場合は、デフォルトの **Open Server** イベント・ハンドラが呼び出されます。
- ユーザ定義のイベントの場合は、トリガ可能となる前に、まず `srv_define_event` で定義する必要があります。
- `srv_event` は、**SRV\_STOP** または **SRV\_START** 以外のイベントをスレッドのイベント・キューに追加します。**SRV\_STOP** または **SRV\_START** イベントの場合、*spp* は、イベントを要求しているスレッドの内部スレッド制御構造体を指します。
- **Open Server** アプリケーションは、I/O として機能するルーチンをユーザ定義イベント内からは呼び出せません。

参照

[srv\\_define\\_event](#), [srv\\_event](#), [srv\\_handle](#), [srv\\_thread\\_props](#), 「イベント」  
(100 ページ)

## srv\_free

説明

割り付けられているメモリを解放します。

構文

```
CS_RETCODE srv_free(mp)
CS_VOID      *mp;
```

パラメータ

*mp*  
解放するメモリを指すポインタです。

戻り値

**表 3-48 : 戻り値 (srv\_free)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE ex_srv_free PROTOTYPE((
CS_BYTE *p
));
/*
```

```

** EX_SRV_FREE
**
** Example routine to free memory allocated through srv_alloc.
**
** Arguments:
**     p - The address of the memory block to be freed.
**
** Returns:
**
**     CS_SUCCEEDED      Memory was freed successfully.
**     CS_FAIL           An error was detected.
*/
CS_RETCODE      ex_srv_free(p)
CS_BYTE        *p;
{
    /*
    ** Free the memory block.
    */
    if( srv_free(p) == CS_FAIL )
    {
        return CS_FAIL;
    }
    return CS_SUCCEEDED;
}

```

#### 使用法

- `srv_free` は、`srv_alloc`、`srv_init`、または `srv_realloc` で割り付けられたメモリを解放する場合にのみ使用します。
- 現行バージョンでは、`srv_free` は C ルーチンである `free` を呼び出します。しかし、Open Server アプリケーションでは、`srv_props` ルーチンを使用して独自のメモリ管理ルーチンをインストールすることができます。ユーザ・インストールのルーチンのパラメータ転送規則は、`free` のものと同一でなければなりません。ユーザ・インストールのルーチンを使用できるようにアプリケーションが設定されていない場合は、`free` を使用してください。

#### 参照

[srv\\_alloc](#), [srv\\_props](#), [srv\\_realloc](#), [srv\\_init](#)

## srv\_freeserveraddrs

**説明** `srv_getserverbyname` によって割り付けられたメモリを解放します。

**構文** `CS_RETCODE srv_freeserveraddrs(void *resultptr)`

パラメータ *resultptr*  
*srv\_getserverbyname* によって返されるメモリへのポインタです。

戻り値 **表 3-49 : 戻り値 (srv\_freesserveraddrs)**

戻り値	意味
CS_SUCCEED	<i>srv_freesserveraddrs</i> の呼び出しを正常に実行した。
CS_FAIL	<i>resultptr</i> が NULL であるか、割り付け解除に失敗した。

参照 [srv\\_getserverbyname](#), [srv\\_send\\_ctlinfo](#)

## srv\_get\_data

説明 *text*、*unitext*、または *image* パラメータ・ストリームをまとまった単位でクライアントから読み込みます。

構文 `CS_RETCODE srv_get_data(spp, bp, buflen, outlenp)`

```
SRV_PROC    *spp;
CS_BYTE     *bp;
CS_INT      buflen;
CS_INT      *outlenp;
```

パラメータ *spp*  
 内部スレッド制御構造体へのポインタです。

*bp*  
 クライアントからのデータが格納されているバッファへのポインタです。

*buflen*  
 それぞれのチャンクで何バイトが転送されたかを示す *\*bp* ポインタのサイズです。

*outlenp*  
*\*bp* バッファに読み込まれたバイト数が、このパラメータに返されます。



戻り値

表 3-50 : 戻り値 (srv\_get\_data)

戻り値	意味
CS_SUCCEED	srv_get_data が正常に実行されました。保留中のデータがさらにあります。
CS_FAIL	ルーチンが失敗。
CS_END_DATA	srv_get_data が text、unitext、または image パラメータ全体の読取を完了しました。

## srv\_get\_text

説明

連続したデータとして、text または image のデータ・ストリームをクライアントから読み込みます。

構文

```
CS_RETCODE srv_get_text(spp, bp, buflen, outlenp)
```

```
SRV_PROC    *spp;
CS_BYTE     *bp;
CS_INT      buflen;
CS_INT      *outlenp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*bp*

クライアントからのデータが格納されているバッファへのポインタです。

*buflen*

\**bp* ポインタのサイズです。このサイズは、連続して転送される 1 つのデータのまとまりをバイト数で示したものです。

*outlenp*

\**bp* バッファに読み込まれたバイト数が、このパラメータに返されます。

戻り値

**表 3-51 : 戻り値 (srv\_get\_text)**

戻り値	意味
CS_SUCCEED	srv_get_text の呼び出しを正常に実行した。
CS_FAIL	ルーチンが失敗。
CS_END_DATA	Open Server が text または image データ・ストリーム全体を読み込んだ。

例

```
#include <ospublic.h>
#include <stdio.h>
/*
** Local Prototype
**/
CS_RETCODE ex_srv_get_text PROTOTYPE((
SRV_PROC *spp,
CS_INT *outlenp,
CS_BYTE *bbuf
));
/*
** EX_SRV_GET_TEXT
**
** Example routine to read chunks of text or image datastream
** from a client into a buffer and then write it to a disk
** file.
**
** Arguments:
**
** spp Pointer to thread control structure.
** outlenp Number of bytes read and written.
** bbuf Pointer to very large buffer for text.
**
** Returns
**
** CS_SUCCEED The data was successfully read.
** CS_FAIL An error was detected.
**
**/
#define BUFSIZE 256
#define FPUTS(a,b) fputs(a,b)
CS_RETCODE ex_srv_get_text(spp,outlenp,bbuf)
SRV_PROC *spp;
CS_INT *outlenp;
CS_BYTE *bbuf;
{
    CS_INT llen; /* Local length.*/
```

```

CS_INT      lout;      /* Local read count.*/
CS_RETCODE  lret;     /* Local return code.*/
CS_BYTE     *lbufp;   /* Local pointer into bbuf.*/
/* Check arguments.*/
if(bbuf == (CS_VOID *)0)
    return(CS_FAIL);
if(spp == (SRV_PROC *)0)
    return(CS_FAIL);
llen = BUFSIZE;
lbufp = bbuf;
/*
** Loop around getting data and copy it to bbuf.
*/
while(lret != CS_END_DATA)
{
    (CS_VOID)srv_bzero(lbufp,BUFSIZE);
    lout = 0;
    lret = srv_get_text(spp, lbufp, llen, &lout);
    if(lret == CS_FAIL)
        break;
    *outlenp += lout;
    lbufp += lout;
}
if(lret == CS_END_DATA)
    return (CS_SUCCEED);
else
    return(lret);
}

```

#### 使用法

- `srv_get_text` は、クライアントからバルク・データを読むために使用します。バルク・データは、`text` や `image` でも可能です。
- クライアントからのバルク・データがすべて読み込まれるまで、`srv_get_text` を呼び出し続けなければなりません。データ・ストリーム全体が読み込まれると、`CS_END_DATA` を返します。
- `srv_get_text` は、`SRV_BULK` イベント・ハンドラからのみ呼び出すことができます。
- `srv_get_text` で読み込まれたカラムは、`text` または `image` でなければなりません。
- `Open Server` アプリケーションは、データ・ストリームに対して初めて `srv_get_text` を呼び出す前に、`srv_text_info` を呼び出す必要があります。次に、アプリケーションは、データ・ストリームを取得するために `srv_get_text` を呼び出します。カラム全体を読み込むために必要な回数だけ、`srv_get_text` が呼び出されます。

- Open Server は、text と image のデータ・ストリームを同等に扱いますが、例外として、text データだけを Open Server アプリケーションに送る前に変換します。Open Server が行う唯一の変換は、文字セット変換です。

参照

[srv\\_bind](#), [srv\\_descfmt](#), [srv\\_send\\_text](#), [srv\\_text\\_info](#), [srv\\_thread\\_props](#), [srv\\_xferdata](#), 「国際化サポート」(111 ページ), 「text と image」(225 ページ)

## srv\_getloginfo

**説明** リモート・サーバとのパススルー接続の準備のために、クライアント・スレッドからログイン情報を取得します。

**構文** CS\_RETCODE `srv_getloginfo(spp, loginfo)`

```
SRV_PROC      *spp;
CS_LOGINFO    **loginfo;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*loginfo*

新しく割り当てられた CS\_LOGINFO 構造体のアドレスに設定される CS\_LOGINFO ポインタへのポインタです。

**戻り値**

**表 3-52 : 戻り値 (srv\_getloginfo)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include      <ospublic.h>
/*
** Local Prototype
*/
extern CS_RETCODE  ex_srv_getloginfo PROTOTYPE((
CS_VOID          *spp,
CS_VOID          **loginfopp
));
/*
** EX_SRV_GETLOGINFO
**
```

```

**      Example routine to retrieve the client' s login structure.
**
** Arguments:
**      spp          Thread control structure.
**      loginfopp   A pointer to client' s login record returned here.
**
** Returns:
**
**      CS_SUCCEEDED   Login structure was retrieved successfully.
**      CS_FAIL        An error was detected.
*/
CS_RETCODE ex_srv_getloginfo(spp, loginfopp)
SRV_PROC          *spp;
CS_LOGININFO **loginfopp;
{
    /* Initialization.*/
    *loginfopp = (CS_LOGININFO *)NULL;
    if(srv_getloginfo(spp, loginfopp) == CS_FAIL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

#### 使用法

- `srv_getloginfo` は、パススルー・モードを使うゲートウェイ・アプリケーションで使用します。パススルー・モードにおいては、ゲートウェイ・アプリケーションはプロトコルを解釈することなく、クライアントとリモート Sybase サーバ間でパケットを相互に渡します。
- クライアントが直接サーバに接続する場合は、2つのプログラムは、データの送受信に使用するプロトコル・フォーマットをネゴシエートします。ゲートウェイ・アプリケーションでプロトコル・パススルーを使用すると、Open Server は、クライアントとリモート・サーバ間でプロトコル・パケットを相互に渡します。つまり、クライアントとリモート・サーバは、プロトコル・バージョンについては一致していなければなりません。
- `srv_getloginfo` は、クライアントとリモート・サーバ間のプロトコル・フォーマットのネゴシエーションを可能にする4つの呼び出しの最初のもので、(そのうち2つは CS-Library 呼び出しです)。これらの呼び出しは、SRV\_CONNECT イベント・ハンドラにおいてのみ行えます。次にその呼び出しを示します。
  - a `srv_getloginfo` - CS\_LOGININFO 構造体を割り付け、クライアント・スレッドからのプロトコル情報を取り込みます。

- b `ct_setloginfo` - 手順 1 で取得したプロトコル情報を使用して `CS_LOGINFO` 構造体を準備し、`ct_connect` を使用してリモート・サーバにログインします。
- c `ct_getloginfo` - `CS_CONNECTION` 構造体から、新しく割り付けられた `CS_LOGINFO` 構造体にプロトコル・ログイン応答情報を転送します。
- d `srv_setloginfo` - 手順 3 で取得したリモート・サーバの応答をクライアントに送信し、`CS_LOGINFO` 構造体を解放します。

参照

[srv\\_recvpassthru](#), [srv\\_sendpassthru](#), [srv\\_setloginfo](#)

## srv\_getmsgq

説明

メッセージ・キューから次のメッセージを取得します。

構文

```
CS_RETCODE srv_getmsgq(msgqid, msgp, getflags, infop)
```

```
SRV_OBJID  msgqid;
CS_VOID    **msgp;
CS_INT     getflags;
CS_INT     *infop;
```

パラメータ

*msgqid*

メッセージを取得するためのメッセージ・キューの識別子です。メッセージ・キューを名前参照するには、メッセージ・キュー ID を取得するための名前を使って `srv_getobjid` を呼び出します。

*msgp*

`srv_getmsgq` がメッセージのアドレスに設定するポインタ変数へのポインタです。

*getflags*

*getflags* の値は、論理和をとることができます。表 3-53 に、*getflags* の有効値とその意味を示します。

表 3-53 : getflags の値 (srv\_getmsgq)

値	説明
SRV_M_WAIT	メッセージがなければ、メッセージが届けられるまで <code>srv_getmsgq</code> はスリープする。
SRV_M_NOWAIT	メッセージがあるないに関係なく、 <code>srv_getmsgq</code> は即座に戻る。
SRV_M_READ_ONLY	<code>srv_getmsgq</code> のデフォルト動作は、メッセージ・リストからメッセージを削除し、そのメッセージが読み込まれることを待っているスレッドがあればそれをウェイクアップする。 <b>SRV_M_READ_ONLY</b> が設定された場合は、メッセージ・ポインタは返されるがメッセージはリストからは削除されず、メッセージを待っているスレッドはウェイクアップしない。このオプションは、メッセージ・キューの先頭を確認して、メッセージがそのスレッドのためのものかを判断することに使用する。

*infp*

`CS_INT` を指すポインタです。表 3-54 に、`srv_getmsgq` が `CS_FAIL` を返す場合に *\*infp* に返される可能性がある値を示します。

表 3-54 : infop の値 (srv\_getmsgq)

値	意味
SRV_I_WOULDWAIT	<code>SRV_M_NOWAIT</code> フラグが <code>getflags</code> フィールドに設定されて、読み込むべきメッセージがない。
SRV_I_DELETED	メッセージを待っている間に、メッセージ・キューが削除された。
SRV_I_INTERRUPTED	<code>SRV_M_WAIT</code> フラグが <code>getflags</code> フィールドに設定され、この呼び出しはメッセージが到着する前に中断された。
SRV_I_UNKNOWN	その他のエラーが発生した。メッセージはログ・ファイルを確認する。

## 戻り値

表 3-55 : 戻り値 (srv\_getmsgq)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <ospublic.h>
/*
** Local prototype
```

```

*/
CS_VOID    ex_srv_getmsgq PROTOTYPE((
SRV_OBJID  msgqid,
CS_INT     *infop
));
/*
** EX_SRV_GETMSGQ
**
** Example routine to get messages from a message queue.
**
** Arguments:
** msgqid-    The id of the message queue from which to get
**            the message.
**
** infop-     Will hold information about why this routine
**            failed.Comes directly from srv_getmsg.
** Returns:
**            Nothing.If this routine returns, it is because srv_getmsgq
**            failed.Check infop to see why it failed.
*/
CS_VOID    ex_srv_getmsgq(msgqid, infop)
SRV_OBJID  msgqid;
CS_INT     *infop;
{
    CS_CHAR  *message; /* This message is a string.*/
    /*
    ** Loop processing messages.Go to sleep if no messages are
    ** available.
    */
    while (srv_getmsgq(msgqid, (CS_VOID *)&message, SRV_M_WAIT,
        infop)== CS_SUCCEEDED)
    {
        /* Process message.*/
    }
    /* infop will contain the reason why it failed.*/
    return;
}

```

**使用方法**

- `srv_getmsgq` は、メッセージ・キュー `msgqid` からの次のメッセージのアドレスを `*msgp` に置きます。
- メッセージを送信したスレッドが、メッセージが読み込まれるまでスリープするように指定している場合、そのスレッドはウェイクアップします。

**参照**

[srv\\_createmsgq](#), [srv\\_deletemsgq](#), [srv\\_getobjid](#), [srv\\_putmsgq](#)



## srv\_getobjid

**説明** 指定の名前を持つメッセージ・キューまたはミューテックスのオブジェクト ID を調べます。

**構文**

```
CS_RETCODE srv_getobjid(obj_type, obj_namep,
                        obj_namelen, obj_idp, infop)

CS_INT      obj_type;
CS_CHAR     *obj_namep;
CS_INT      obj_namelen;
SRV_OBJID   *obj_idp;
CS_INT      *infop;
```

**パラメータ**

*obj\_type*

オブジェクトがミューテックス (SRV\_C\_MUTEX) またはメッセージ・キュー (SRV\_C\_MQUEUE) のどちらであるかを示します。

*obj\_namep*

オブジェクトの名前が格納されている CS\_CHAR バッファへのポインタです。

*obj\_namelen*

\**obj\_namep* の文字列の長さです。文字列が null で終了する場合、*obj\_namelen* は CS\_NULLTERM とすることもできます。

*obj\_idp*

オブジェクトの識別子を受け取る SRV\_OBJID 構造体へのポインタです。

*infop*

CS\_INT を指すポインタです。表 3-56 に、*srv\_getobjid* が CS\_FAIL を返す場合に \**infop* に返される可能性がある値を示します。

**表 3-56 : infop の値 (srv\_getobjid)**

値	意味
SRV_I_NOEXIST	オブジェクトは存在しない。
SRV_I_UNKNOWN	null オブジェクト名など、他のエラーが生じた。

**戻り値**

**表 3-57 : 戻り値 (srv\_getobjid)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include <ospublic.h>
```

```

/*
** Local Prototype
*/
CS_INT    ex_srv_getobjid PROTOTYPE((
CS_INT    obj_type,
CS_CHAR   *obj_name,
SRV_OBJID *obj_idp
));
/*
** EX_SRV_GETOBJID
** An example routine to retrieve the object id for a specified
** message queue or mutex name.
** Arguments:
** obj_type  SRV_C_MUTEX if requesting a mutex object id, and
**            SRV_C_MQUEUE if requesting a message queue object
**            id.
** obj_name  A null terminated string which specifies the name
**            of the message queue or the mutex.
** obj_idp   A pointer to a SRV_OBJID structure that will store
**            the identifier for the object.
** Returns:
** CS_SUCCEED   If the object id was retrieved
**                successfully.
** SRV_I_NOEXIST If the object does not exist.
** CS_FAIL      If the object was not retrieved due to an error
*/
CS_INT    ex_srv_getobjid(obj_type, obj_name, obj_idp)
CS_INT    obj_type;
CS_CHAR   *obj_name;
SRV_OBJID *obj_idp;
{
    CS_INT    info; /* The reason for failure.*/
    CS_INT    status; /* The return status.*/
    /* Validate the obj_type.*/
    if ( (obj_type != SRV_C_MUTEX) && (obj_type !=
        SRV_C_MQUEUE) )
    {
        return(CS_FAIL);
    }
    /* Make sure that the object name is not null.*/
    if ( obj_name == (CS_CHAR *)NULL )
    {
        return(CS_FAIL);
    }
    /* Ensure that the pointer to the SRV_OBJID is not null */
    if ( obj_idp == (SRV_OBJID *)NULL )

```

```

{
    return(CS_FAIL);
}
/* Get the object id. */
status = (CS_INT)srv_getobjid( obj_type, obj_name,
                             CS_NULLTERM, obj_idp, &info);
/* Check the status.*/
if ( (status == CS_FAIL) && (info == SRV_I_NOEXIST) )
{
    status = SRV_I_NOEXIST;
}
return(status);
}

```

**使用法** Open Server は、メッセージ・キューやミューテックスのユニークなオブジェクト識別子をそれぞれの名前にマップするテーブルを保持しています。名前が与えられると、`srv_getobjid` はその識別子を探します。

**参照** [srv\\_createmsgq](#), [srv\\_createmutex](#), [srv\\_deletemsgq](#), [srv\\_deletemutex](#), [srv\\_getmsgq](#), [srv\\_getobjname](#), [srv\\_lockmutex](#), [srv\\_putmsgq](#), [srv\\_unlockmutex](#)

## srv\_getobjname

**説明** 指定した識別子を持つ、メッセージ・キューまたはミューテックスの名前を取得します。

**構文** `CS_RETCODE srv_getobjname(obj_type, obj_id, obj_namep, obj_namelenp, infop)`

```

CS_INT      obj_type;
SRV_OBJID   obj_id;
CS_CHAR     *obj_namep;
CS_INT      *obj_namelenp;
CS_INT      *infop;

```

**パラメータ** `obj_type`  
オブジェクトがミューテックス (`SRV_C_Mutex`) またはメッセージ・キュー (`SRV_C_MQueue`) のどちらであることを示します。

`obj_id`  
オブジェクトのユニークな識別子です。

*obj\_namep*

オブジェクトの名前がコピーされている CS\_CHAR バッファへのポインタです。バッファのサイズは、オブジェクト名、および null 文字 (*obj\_namelenp* が NULL の場合) を格納できる大きさでなければなりません。オブジェクト名の最大長は、null 終了バイトを含めない SRV\_MAXNAME の文字の長さです。

*obj\_namelenp*

オブジェクトの長さを受け取る CS\_INT へのポインタです。*obj\_namelenp* が NULL の場合、検索された名前は *\*obj\_namep* にコピーされ、NULL 文字で終了します。NULL 以外の場合は、*\*obj\_namep* の名前の長さは *\*obj\_namelenp* に格納されます。

*infop*

ID *obj\_id* のオブジェクトが存在しない場合に、SRV\_I\_NOEXIST に設定される CS\_INT へのポインタです。

戻り値

**表 3-58 : 戻り値 (srv\_getobjname)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
#include <stdio.h>
/*
** Local Prototype
*/
CS_RETCODE ex_srv_getobjname PROTOTYPE((
CS_INT obj_type,
SRV_OBJID obj_id
));
/*
** EX_SRV_GETOBJNAME
** Example routine to illustrate the use of srv_getobjname to
** get the name of mutex or message queue with id = obj_id
** where obj_id was earlier returned by srv_createmutex or
** srv_createmsgq.
** Arguments:
** obj_type - Type of object; SRV_C_MUTEX or SRV_C_MQUEUE.
** obj_id - The unique identifier of the object.
** Returns:
** CS_SUCCEED Memory was allocated successfully.
** CS_FAIL Memory allocation failure occurred.
*/
```

```

CS_RETCODE   ex_srv_getobjname(obj_type, obj_id)
CS_INT       obj_type;
SRV_OBJID    obj_id;
{
    CS_CHAR    obj_name[SRV_MAXNAME+1];
    CS_INT     obj_namelen;
    CS_INT     info;
    CS_RETCODE ret;
    /* Get object name.*/
    ret = srv_getobjname(obj_type, obj_id, obj_name,
        &obj_namelen, &info);
    /* Print information depending on retcode */
    switch(ret)
    {
        case CS_FAIL:
            if (info == SRV_I_NOEXIST)
            {
                fprintf(stderr, "%s object with id:%d does not
                    exist\n", (obj_type == SRV_C_Mutex) ?
                    "Mutex" : "Message Queue", (CS_INT)obj_id);
            }
            else
                fprintf (stderr, "srv_getobjname failed\n");
            break;
        case CS_SUCCEED:
            fprintf (stderr, "%s name:%s for id:%d\n",
                (obj_type == SRV_C_Mutex) ? "Mutex" : "Message Queue",
                obj_name, (CS_INT)obj_id);
            break;
        default:
            fprintf (stderr, "Unknown return code from
                srv_getobjname\n");
            ret = CS_FAIL;
            break;
    }
    return (ret);
}

```

#### 使用法

- Open Server では、メッセージ・キューやミューテックスのユニークな識別子をメッセージ・キューやミューテックスの名前にマップするテーブルを保持しています。識別子を指定すると、`srv_getobjname` によって名前が検索されます。

- アプリケーションによっては、リファレンス・メッセージ・キューやミューテックスを、名前で調べた方が便利な場合もあります。ミューテックスやメッセージ・キュー・サービスを使用している識別子を調べるために、`srv_getobjid`を使用することもできます。

参照 [srv\\_createmsgq](#), [srv\\_createmutex](#), [srv\\_deletemsgq](#), [srv\\_deletemutex](#), [srv\\_getmsgq](#), [srv\\_getobjid](#), [srv\\_lockmutex](#), [srv\\_putmsgq](#), [srv\\_unlockmutex](#)

## srv\_getserverbyname

**説明** `server_name` の接続情報を返し、必要に応じてメモリを割り付けます。`srv_getserverbyname` によって割り付けたメモリは、`srv_freesserveraddrs` を呼び出すことで解放できます。

**構文** `CS_RETCODE srv_getserverbyname(CS_CHAR *server_name, CS_INT namelen, CS_INT querytype, CS_INT result_type, void *resultptr, CS_INT *result_cnt)`

**パラメータ**

`server_name`  
 検索するサーバの名前です。

`namelen`  
`server_name` の長さ。CS\_NULLTERM として指定できます。

`querytype`  
`server_name` に対してマスタ (CS\_ACCESS\_CLIENT\_MASTER) またはクエリ (CS\_ACCESS\_CLIENT\_QUERY) のエントリを選択します。

`result_type`  
 接続情報のデータ・フォーマットを示します。`result_type` は、SRV\_C\_GETADDRS または SRV\_C\_GETSTRS として指定できます。

`resultptr`  
 クエリの結果を保持するために `srv_getserverbyname` によって割り付けられるポインタです。`resultptr` は、クエリ結果のアドレスを受け取るポインタのアドレスです。

`result_cnt`  
`server_name` に対して返されるアドレスの数を含む CS\_INT へのポインタです。

使用法	<p><i>result_type</i> は <code>SRV_C_GETADDRS</code> として指定できます。情報は <code>CS_TRANADDR</code> 構造体の配列として返されます。または、<i>result_type</i> を <code>SRV_C_GETSTRS</code> として指定すると、文字列へのポインタの配列を <i>network-protocol protocol-address filter-information</i> というフォーマットで返すことができます。たとえば、<i>network-protocol</i> が “tcp”、<i>protocol-address</i> が “myhost 4000”、<i>filter-information</i> が “ssl” である場合は、“tcp myhost 4000 ssl” の結果を受け取ります。</p>
参照	<a href="#">srv_freeserveraddrs</a> , <a href="#">srv_send_ctlinfo</a>

## srv\_handle

説明	Open Server アプリケーションにイベント・ハンドラをインストールします。
構文	<pre>SRV_EVENTHANDLE_FUNC (*srv_handle(ssp, event,                                    handler))()  SRV_SERVER             *ssp; CS_INT                 event; SRV_EVENTHANDLE_FUNC  handler;</pre>
パラメータ	<p><i>ssp</i></p> <p>Open Server の制御構造体へのポインタです。このパラメータはオプションです。下位互換性を提供します。</p>

*event*

*handler* が処理するイベントです。通常の Open Server イベントのリストを次に示します。

- SRV\_ATTENTION
- SRV\_BULK
- SRV\_CONNECT
- SRV\_CURSOR
- SRV\_DISCONNECT/SRV\_URGDISCONNECT
- SRV\_DYNAMIC
- SRV\_FULLPASSTHRU
- SRV\_LANGUAGE
- SRV\_MSG
- SRV\_OPTION
- SRV\_RPC
- SRV\_START
- SRV\_STOP

ユーザ定義のイベント – ユーザ定義のイベントは、`srv_define_event` を使用して定義されます。

各イベントの詳細については、「[イベント](#)」(100 ページ) を参照してください。

ハンドラ

*event* 要求が生じたときに呼び出す関数へのポインタです。ハンドラとして NULL を渡すことによって、デフォルトのイベント・ハンドラがインストールされます。

戻り値

**表 3-59 : 戻り値 (srv\_handle)**

戻り値	意味
イベント処理関数へのポインタ	関数のロケーション。
null ポインタ	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local Prototype
*/
```



```

extern CS_RETCODE  ex_srv_handle PROTOTYPE((
SRV_EVENTHANDLE_FUNC  funcp
));
/*
** EX_SRV_HANDLE
** Install a SRV_START handler.
** Arguments:
**  funcp  Handler to install.
** Returns:
**  CS_SUCCEED  Start handler was installed successfully.
**  CS_FAIL    An error was detected.
*/
CS_RETCODE          ex_srv_handle(funcp)
SRV_EVENTHANDLE_FUNC  funcp;
{
    if(srv_handle((SRV_SERVER *)NULL, SRV_START, funcp) ==
        CS_FAIL )
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

#### 使用法

- `srv_handle` は、Open Server が特定のイベントの処理要求を受けた場合、Open Server に特定の関数を呼び出すように通知します。
- Open Server は、1つの引数を使って *handler* を呼び出します。  
 下記のイベントの場合、イベント・ハンドラの引数として、Open Server 制御構造体へのポインタが必要です。
  - SRV\_START
  - SRV\_STOP
 下記のイベントの場合、イベント・ハンドラの引数として、スレッド制御構造体へのポインタが必要です。
  - SRV\_ATTENTION
  - SRV\_BULK
  - SRV\_CONNECT
  - SRV\_CURSOR
  - SRV\_DISCONNECT/SRV\_URGDISCONNECT
  - SRV\_DYNAMIC
  - SRV\_FULLPASSTHRU

- SRV\_LANGUAGE
- SRV\_MSG
- SRV\_OPTION
- SRV\_RPC

あらゆるユーザ定義のイベント

- 各 Open Server イベントには、既知の名前を持つデフォルト・ハンドラがあります。`srv_handle` でイベント・ハンドラをインストールすると、デフォルト・ハンドラが置換されます。
- イベント・ハンドラは、動的にインストールすることができます。次のイベント発生時に、新しいイベント・ハンドラが呼び出されます。
- イベント・ハンドラは、`CS_SUCCEED` を返す必要があります。

参照

[srv\\_define\\_event](#), [srv\\_event](#), [srv\\_event\\_deferred](#), 「イベント」 (100 ページ)

## srv\_init

説明

Open Server アプリケーションを初期化します。

構文

```
SRV_SERVER *srv_init(scp, servernamep, namelen)
SRV_CONFIG *scp;
CS_CHAR *servernamep;
CS_INT namelen;
```

パラメータ

*scp*

すべての Open Server 設定オプションの値を保持する設定構造体です。この引数はオプションであり、下位互換性を提供します。

*servernamep*

Open Server アプリケーション名へのポインタです。指定した名前は、必要なネットワーク情報を取得するために `interfaces` ファイル内で検索されます。Open Server の名前として (`CS_CHAR *`) `NULL` を使用する場合は、`DSLISTEN` の値がサーバ名になります。

`DSLISTEN` が明示的に設定されていない場合には、デフォルトの文字列 “SYBASE” がサーバ名になります。

*namelen*

*\*servernamep* の文字列の長さをバイト数で示したものです。文字列が (CS\_CHAR \*) NULL の場合は、*namelen* は無視されます。文字列が null で終了する場合、*namelen* は CS\_NULLTERM とすることもできます。

戻り値

表 3-60 : 戻り値 (srv\_init)

戻り値	意味
SRV_SERVER ポインタ	ルーチンが正常に実行された。
(SRV_SERVER *) NULL	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local prototype.
*/
SRV_SERVER      *ex_srv_init PROTOTYPE((
SRV_CONFIG      *scp
));
/*
** EX_SRV_INIT
**
** Example routine to initialize an Open Server application.
**
** Arguments:
** scp - A pointer to the configuration structure.
**
** Returns:
** On success, a pointer to a newly allocated SRV_SERVER
** structure.
** On failure, NULL.
**
**/
SRV_SERVER      *ex_srv_init(scp)
SRV_CONFIG      *scp;
{
    SRV_SERVER      *server;
    CS_CHAR          *servername = "EX_SERVER" ;
    server = srv_init(scp, servername, CS_NULLTERM);
    return (server);
}
```

使用法

- サーバは初期化してから *srv\_run* で起動します。

- `srv_init` は Open Server アプリケーションを初期化します。初期化の過程を構成する主なものは、サーバに必要なデータ構造体の割り付け、サーバ・ステータスの初期化、およびネットワーク・リスナの起動です。
- デフォルト以外の値を指定する場合は、`srv_init` を呼び出す前に設定オプションを設定する必要があります。設定可能なオプションのリストについては、[srv\\_props](#) のページを参照してください。
- ライブラリのバージョン情報とデフォルト国際化値を設定するには、`srv_version` を呼び出してから `srv_init` を呼び出します。
- `SRV_STOP` イベントが発生すると、Open Server は `SRV_SERVER` 構造体を解放します。`SRV_SERVER` 構造体を Open Server アプリケーションが解放しないようにしてください。
- `interfaces` ファイルの指定に関する情報については、[srv\\_props](#) リファレンス・ページを参照してください。詳細については、使用しているプラットフォームの『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

参照 [srv\\_props](#), [srv\\_run](#), [srv\\_version](#)

## srv\_langcpy

**説明** クライアントの言語要求を、アプリケーション・バッファにコピーします。

**構文** `CS_INT srv_langcpy(spp, start, nbytes, bp)`

```
SRV_PROC *spp;
CS_INT start;
CS_INT nbytes;
CS_BYTE *bp;
```

**パラメータ** *spp*  
内部スレッド制御構造体へのポインタです。

*start*  
要求バッファの文字のコピーを開始する位置です。要求バッファの最初の文字の位置は、0 番目となります。

*nbytes*

コピーする文字数です。*nbytes* が -1 の場合、*srv\_langcpy* は可能なかぎりの文字をコピーします。0 バイトをコピーすることもできます。コピーする文字が *nbytes* に満たない場合は、*srv\_langcpy* は要求バッファにあるだけの文字をコピーします。

*bp*

バイトのコピー先となるユーザ提供バッファへの *CS\_CHAR* ポインタです。

## 戻り値

表 3-61 : 戻り値 (*srv\_langcpy*)

戻り値	意味
整数	コピーしたバイト数。
-1	このクライアントから現在の言語要求はない。

## 例

```
#include      <ospublic.h>

/*
** Local Prototype
*/
CS_RETCODE    ex_srv_langcpy PROTOTYPE((
SRV_PROC      *spp,
CS_CHAR       *buf,
CS_INT        size,
CS_INT        *outlen
));

/*
** EX_SRV_LANGCPY
**
** Example routine to illustrate the use of srv_langcpy to
** copy language commands sent by a client.
**
** Arguments:
** spp      A pointer to internal thread control structure.
** buf      A CS_CHAR pointer to buffer for language commands.
** size     The size of the buffer; A CS_INT.
** outlen   A pointer to CS_INT; the actual length of
**           language query copied to buf is returned here.-1
**           is returned in case of failure.
**
** Returns:
**
** CS_SUCCEEDED Language request was copied successfully.
```

```

**      CS_FAIL          An error was detected.
*/
CS_RETCODE  ex_srv_langcpy(spp, buf, size, outlen)
SRV_PROC    spp;
CS_CHAR     *buf;
CS_INT      size;
CS_INT      *outlen;
{
    CS_INT    act_len; /* actual length of language request */

    /* Initialization.*/
    *outlen = (CS_INT)-1;

    /* Get the length of language request.*/
    if ((act_len = srv_langlen(spp)) == -1)
        return(CS_FAIL);

    /* Check to see whether we got a buffer of adequate size.*/
    if (size < (act_len + 1))
        return(CS_FAIL);

    /* Copy language commands.*/
    if (srv_langcpy(spp, (CS_INT)0, act_len, buf) <= 0)
        return(CS_FAIL);

    /* Set the actual length copied.*/
    *outlen = act_len;

    return (CS_SUCCEED);
}

```

#### 使用法

- クライアントから言語要求を受信すると、`srv_langcpy` は要求バッファの一部を **Open Server** プログラム変数にコピーできます。送信先バッファに格納されたコピーは、`null` で終了します。
- `srv_langcpy` は、カーソル宣言または更新文において言語文字列の処理にも使用されます。

---

**警告！** `srv_langcpy` は、送信先バッファが最低でも `nbytes + 1` のバイト数を処理できるサイズがあることを想定します。

---

- 言語要求バッファの全体の長さを変更するには、`srv_langlen` を呼び出します。

- 要求バッファには、Transact-SQL 文だけでなく、どのような文字列でも格納できます。文字列の処理は、Open Server アプリケーションの責任です。

参照 [srv\\_langlen](#)

## srv\_langlen

説明 言語要求バッファの長さを返します。

構文  
 CS\_INT srv\_langlen(spp)  
 SRV\_PROC \*spp;

パラメータ *spp*  
 内部スレッド制御構造体へのポインタです。

戻り値 **表 3-62 : 戻り値 (srv\_langlen)**

戻り値	意味
整数	言語要求バッファの長さをバイト数で示したものの。
-1	このクライアントから現在の言語要求はない。

例

```
#include <ospublic.h>
/*
** Local Prototype
**
CS_RETCODE ex_srv_langlen PROTOTYPE((
SRV_PROC *spp,
CS_INT *len
));

/*
** EX_SRV_LANGLEN
** Example routine to return the length of the language request
** buffer using srv_langlen.
**
** Arguments:
** spp A pointer to the internal thread control structure.
** len Return pointer for the length of the language string.
** If there is no language command -1 is returned.
**
**
```

```

** Returns:
**
**   CS_SUCCEED   Language length was retrieved successfully.
**   CS_FAIL     An error was detected.
**
**/
CS_RETCODE   ex_srv_langlen(spp, len)
SRV_PROC     *spp;
CS_INT       *len;
{
    /* Retrieve the language length.*/
    if ((*len = srv_langlen(spp)) < 0)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

使用法

- クライアントから言語要求を受信すると、`srv_langlen` は要求バッファの長さを返します。
- `srv_langlen` は、カーソル宣言または更新文において言語文字列の処理にも使用されます。
- 要求バッファの全部または一部は、`srv_langcpy` でアクセスできません。
- 要求バッファは、Transact-SQL 文を含む、どのような文字列でも含むことができます。文字列の処理は、Open Server アプリケーションの責任です。

参照

[srv\\_langcpy](#)

## srv\_lockmutex

説明

ミューテックスをロックします。

構文

```

CS_RETCODE srv_lockmutex(mutex_id, waitflag, infop)
SRV_OBJID   mutex_id;
CS_INT      waitflag;
CS_INT      *infop;

```

パラメータ

*mutex\_id*

`srv_createmutex` への呼び出しで返されるユニークなミューテックス識別子です。ミューテックスの名前を指定すると、`srv_getobjid` の呼び出しで *mutex\_id* を取得できます。



*waitflag*

ミューテックスがただちに付与されない場合は、ミューテックス・ロックを要求しているスレッドは待っているか、単に戻るかを指定します。*\*indp* の値は、ロックが付与されたかどうかを示します。*waitflag* の有効値は 2 つあります。ロックがただちに付与されない場合にスレッドが待つことを指定する `SRV_M_WAIT` と、ロックが付与されない場合にスレッドが戻ることを指定する `SRV_M_NOWAIT` です。

*infop*

次の値のいずれかに設定した `CS_INT` へのポインタです。

`SRV_I_SYNC` — ロックは同期的に付与されました。つまり、ロックを要求していたスレッドにロックを待機するための中断はありませんでした。`srv_lockmutex` は `CS_SUCCEED` を返しました。

`SRV_I_GRANTED` — ロックを要求していたスレッドが中断されて、他のスレッドがミューテックスのロックを解放するまで待機した後、ロックは付与されました。`srv_lockmutex` は `CS_SUCCEED` を返しました。

`SRV_I_INTERRUPTED` — スレッドはロックを待機している間にアテンションを受け取りました。ロックは付与されず、`srv_lockmutex` は `CS_FAIL` を返しました。

`SRV_I_WOULDWAIT` — *waitflag* パラメータは `SRV_M_NOWAIT` に設定され、スレッドはロックを待機しなければならない状態でした。ロックは付与されず、`srv_lockmutex` は `CS_FAIL` を返しました。

`SRV_I_UNKNOWN` — ミューテックスが存在しないなど、その他のエラーが発生しました。`srv_lockmutex` は `CS_FAIL` を返しました。

戻り値

表 3-63 : 戻り値 (`srv_lockmutex`)

戻り値	意味
<code>CS_SUCCEED</code>	ルーチンが正常に終了した。
<code>CS_FAIL</code>	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
  ** Local Prototype
  */
CS_RETCODE      ex_srv_lockmutex PROTOTYPE((
SRV_OBJID      mid
));
/*
```

```

** EX_SRV_LOCKMUTEX
**
** Example routine to illustrate the use of srv_lockmutex.
**
** Arguments:
**     mid - The id of the mutex to lock.
**
** Returns:
**
**     CS_SUCCEED   Mutex successfully locked.
**     CS_FAIL      An error was detected.
**
*/
CS_RETCODE   ex_srv_lockmutex(mid)
SRV_OBJID    mid;      /* The mutex id. */
{
    CS_INT    info;     /* Information output variable.*/

    /*
    ** Request the mutex lock - sleep until we get it.
    */
    if( srv_lockmutex(mid, SRV_M_WAIT, &info) == CS_FAIL )
    {
        /*
        ** An error was already raised.
        */
        return CS_FAIL;
    }

    /*
    ** All done.
    */
    return CS_SUCCEED;
}

```

#### 使用法

- ミューテックスは、複数スレッドによる同時アクセスから保護されなければならないデータ・オブジェクトやプログラム・リソースと関連付けられています。
- ミューテックス・ロックは、先着優先で付与されます。
- ロックが付与されるのは、他のスレッドがすでにミューテックスのロックを得ていない場合のみです。
- `srv_lockmutex` は、`SRV_START` または `SRV_ATTENTION` ハンドラでは使用できません。

- スレッドは同じミューテックスを何度もロックすることができませんが、他のスレッドがミューテックスをロックする前に `srv_lockmutex` を呼び出した回数だけ、`srv_unlockmutex` も一度ずつ呼び出す必要があります。
- 待っていたミューテックスが削除された場合、`srv_lockmutex` は `CS_FAIL` を返します。

参照

[srv\\_createmutex](#), [srv\\_deletemutex](#), [srv\\_getobjid](#), [srv\\_unlockmutex](#)

## srv\_log

説明

Open Server ログ・ファイルにメッセージを書き込む。

構文

```
CS_RETCODE srv_log(ssp, datestamp, msgp, msglen)
```

```
SRV_SERVER *ssp;
CS_BOOL    datestamp;
CS_CHAR    *msgp;
CS_INT     msglen;
```

パラメータ

*ssp*

Open Server へのハンドラです。この引数はオプションであり、下位互換性を提供します。

*datestamp*

*datestamp* が `CS_TRUE` の場合、現在の日時がログ・メッセージの先頭に追加されます。*datestamp* が `CS_FALSE` の場合、ログ・メッセージに `timestamp` が設定されません。

*msgp*

メッセージの実際のテキストへのポインタです。

*msglen*

*msg* の長さをバイト数で示したものです。*\*msgp* の文字列が `null` で終了する場合、*msglen* は `CS_NULLTERM` とすることもできます。

戻り値

表 3-64 : 戻り値 (srv\_log)

戻り値	意味
<code>CS_SUCCEED</code>	ルーチンが正常に終了した。
<code>CS_FAIL</code>	ルーチンが失敗。

例

```
#include <ospublic.h>
```

```

#include <string.h>
/*
** Local Prototype.
*/
CS_RETCODE    ex_srv_log PROTOTYPE((
SRV_SERVER    *ssp,
CS_CHAR       *msg_txt
));
/*
** EX_SRV_LOG
**
** Example routine to log a message.
**
** Arguments:
**
**     ssp        A pointer to the Open Server state information
**                control structure.
**     msg_txt    Text of message to log.
** Returns
**
**     CS_SUCCEED      Thread was created.
**     CS_FAIL        An error was detected.
**
**
*/
CS_RETCODE    ex_srv_log(ssp, msg_txt)
SRV_SERVER    *ssp;
CS_CHAR       *msg_txt;
{
    CS_RETCODE    lret;
    CS_INT        msg_len;
    /* Check arguments.*/
    if(ssp == (SRV_SERVER *)0)
        return(CS_FAIL);
    if(msg_txt == (CS_CHAR *)NULL)
        return(CS_FAIL);
    msg_len=strlen(msg_txt);
    /*
    ** Log the message - We use CS_TRUE as the second argument
    **                    to force the date and time to be
    **                    added to the beginning of the logged
    **                    message.If you do not want a
    **                    datestamp then use CS_FALSE.
    */
    lret = srv_log(ssp,CS_TRUE,msg_txt,msg_len);
    return(lret);
}

```

- 使用法
- `srv_log` は、Open Server ログ・ファイルにメッセージを書き込みます。ログ・ファイルのデフォルト名は、`srv.log` です。名前は、`srv_props` で設定できます。
  - メッセージはログ・ファイルに付け加えられます。
  - ログ・ファイルの名前は、`srv_props` ルーチンでアクセスできません。
  - 改行文字は、`*msgp` のテキストには追加されません。
  - ログ・ファイルは、`srv_props` によって設定された `SRV_TRUNCATELOG` プロパティに基づいてトランケートされます。
  - メッセージの長さが `SRV_MAXMSG` を超える場合、Open Server はメッセージをトランケートします。これは、メッセージが `null` で終了しているかどうかに関係ありません。
  - `srv_init` が終了していない場合、メッセージはブート・ウィンドウに表示されます。

参照 [srv\\_props](#)

## srv\_mask

説明 `SRV_MASK_ARRAY` 構造体のビットを初期化、設定、クリア、またはチェックします。

構文 `CS_RETCODE srv_mask(cmd, maskp, bit, infop)`

```
CS_INT          cmd;
SRV_MASK_ARRAY *maskp;
CS_INT          bit;
CS_BOOL         *infop;
```

パラメータ `cmd`  
実行中のアクションです。表 3-65 に、`cmd` の有効値を示します。

**表 3-65 : cmd の有効値 (srv\_mask)**

値	アクション
CS_SET	*maskp の SRV_MASK_ARRAY の bit を設定する。
CS_GET	*maskp の SRV_MASK_ARRAY で bit が現在設定されているかどうかを調べる。bit が設定されている場合は、*infop は CS_TRUE に設定される。そうでない場合は、CS_FALSE に設定される。
CS_CLEAR	*maskp の SRV_MASK_ARRAY の bit をクリアする。
CS_ZERO	すべての bit がオフになるように、*maskp の SRV_MASK_ARRAY を初期化する。cmd が CS_ZERO に設定されている場合は、bit と infop が無視される。

*maskp*

SRV\_MASK\_ARRAY 構造体へのポインタです。

*bit*

SRV\_MASK\_ARRAY で初期化、設定、クリア、またはチェックされているビットです。これは、0 から SRV\_MAXMASK\_LENGTH の間の整数でなければなりません。SRV\_MAXMASK\_LENGTH は、*ospublic.h* で定義されます。

*infop*

*bit* が設定されているかどうかを示す変数へのポインタです。cmd が CS\_SET、CS\_CLEAR、または CS\_ZERO の場合は、このパラメータは無視されます。

戻り値

**表 3-66 : 戻り値 (srv\_mask)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_mask PROTOTYPE((
SRV_MASK_ARRAY *maskptr,
CS_INT bit
));

/*
** EX_SRV_MASK
```

```

**
**      Example routine to manipulate bits in a SRV_MASK_ARRAY
**      structure.
**
** Arguments:
**      maskptr      A pointer to a mask array.
**      bit          The bit to examine.
**
** Returns:
**
**      CS_SUCCEED
**      CS_FAIL
*/
CS_RETCODE      ex_srv_mask(maskptr, bit)
SRV_MASK_ARRAY *maskptr;
CS_INT  bit;
{
    CS_BOOL      info = CS_TRUE;

    if (srv_mask(CS_GET, maskptr, bit, &info) == CS_FAIL)
    {
        return(CS_FAIL);
    }
    else
    {
        /* Has the bit been set?*/
        if (info == CS_FALSE)
            return(CS_FAIL);
        else
            return (CS_SUCCEED);
    }
}

```

**使用法**                      `srv_mask` は、`SRV_MASK_ARRAY` にアクセスし、変更するために使用されます。

## srv\_msg

**説明**                              メッセージ・データ・ストリームを送信または受信します。

**構文**                              `CS_RETCODE srv_msg(spp, cmd, msgidp, status)`

```

SRV_PROC      *spp;
CS_INT        cmd;

```

CS\_INT \*msgidp;  
 CS\_INT \*statusp;

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

アプリケーションがメッセージの送信または取得のどちらを行うために `srv_msg` を呼び出しているかを示します。表 3-67 に、`cmd` の有効値を示します。

**表 3-67 : `cmd` の値 (`srv_msg`)**

値	説明
CS_SET	<code>srv_msg</code> は、メッセージをクライアントに送信する前に <code>status</code> および <code>msgid</code> の値を設定している。
CS_GET	<code>srv_msg</code> は、受信しているメッセージの <code>status</code> および <code>msgid</code> の値を取得している。

*msgidp*

現在のメッセージのメッセージ ID へのポインタです。Open Server アプリケーションでメッセージを送信する場合 (CS\_SET)、ここで ID を指定してください。アプリケーションがメッセージを読み込んでいる場合 (CS\_GET)、受信したメッセージのメッセージ ID がここに返されます。SRV\_MINRESMSG から SRV\_MAXRESMSG の値は、Sybase の内部使用のために予約されています。続いてメッセージ ID が TDS から `smallint` (2 バイト) として送信されるので、メッセージ ID を符号なしの CS\_SMALLINT として定義した場合、ユーザ自身のメッセージに使用できる有効範囲は SRV\_MAXRESMSG から 65535 までです。

*statusp*

現在のメッセージのステータスへのポインタです。Open Server アプリケーションがメッセージを受信している場合 (CS\_GET)、Open Server はメッセージ・ステータスで `*statusp` を更新します。アプリケーションがメッセージを送信している場合 (CS\_SET)、`*statusp` には、送られるメッセージのステータスが含まれていることが必要です。表 3-68 に、`*statusp` の有効値を示します。



表 3-68 : statusp の値 (srv\_msg)

値	説明
SRV_HASPARAMS	メッセージにパラメータがある。
SRV_NOPARAMS	メッセージにパラメータはない。

戻り値

表 3-69 : 戻り値 (srv\_msg)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local prototype.
**
*/
CS_RETCODE      ex_srv_msg PROTOTYPE((
SRV_PROC       *spp
));

/*
** EX_SRV_MSG
**
** Example routine to receive and send a message datastream.
**
** Arguments:
**     spp      A pointer to an internal thread control structure.
**
** Returns:
**     CS_SUCCEED if we were successful in both receiving and
**     sending a message stream.
**
**     CS_FAIL if an error was detected.
**
*/
CS_RETCODE      ex_srv_msg(spp)
SRV_CONFIG      *scp;
{
    CS_RETCODE      result;
    CS_INT          msgid;
    CS_INT          status;

    /*
     ** We will first get a message and process any parameters.
     **
     */

```

```

result = srv_msg(spp, CS_GET, &msgid, &status);

if (result == CS_FAIL)
{
    return(CS_FAIL);
}

if (status == SRV_HASPARAMS)
{
    /*
     ** Process parameters here using srv_bind and
     ** srv_xferdata.
     */
}

/*
** Now, an example of sending a message.
*/
msgid = 32768;
status = SRV_NOPARAMS;

result = srv_msg(spp, CS_SET, &msgid, &status);

if (result == CS_FAIL)
{
    return(CS_FAIL);
}
/*
** If the message has parameters, send it across using      ** srv_xferdata
*/
if (status == SRV_HASPARAMS)
{
    result = srv_xferdata(spp, CS_SET, SRV_MSGDATA);
}
return (result);
}

```

**使用法**

- `srv_msg` は、TDS メッセージ・データ・ストリームの送信または受信に使用されます。
- クライアントから受信するメッセージ・データ・ストリームはすべて、`SRV_MSG` イベントを発生させます。受信された各メッセージに対して個別のイベントが発生します。

- メッセージにパラメータがある場合は、*\*statusp* は `CS_HASPARAMS` の値を含んでいます。アプリケーションは、*type* を `SRV_MSGDATA` に設定して、*srv\_descfmt*、*srv\_bind*、*srv\_xferdata* を使用してパラメータの取得と保存ができます。
- アプリケーションは、メッセージのパラメータの数を *srv\_numparams* を呼び出して調べることができます。
- *srv\_msg* ルーチンは、ステータスと ID を送信するために使用されます。メッセージの実際のパラメータは、パラメータが存在する場合、*type* 引数を `SRV_MSGDATA` に設定して、*srv\_descfmt*、*srv\_bind*、*srv\_xferdata* を使用して送信されます。
- アプリケーションは、複数のメッセージ・データ・ストリームを送受信することができます。
- *srv\_xferdata* は、メッセージ・パラメータの取得または送信の場合にかぎり必要です。これらの目的で使用する場合、送信または受信するメッセージごとに、*srv\_xferdata* を 1 回呼び出してください。パラメータが存在しないときに *srv\_xferdata* を使用すると、Open Server はエラーを返します。
- *srv\_msg* を `SRV_MSG` イベント・ハンドラで呼び出せるのは、*cmd* が `CS_GET` に設定されているときに限ります。*cmd* が `CS_SET` の場合は、どのハンドラでも呼び出すことができます。

## 参照

[srv\\_bind](#)、[srv\\_descfmt](#)、[srv\\_numparams](#)、[srv\\_xferdata](#)、「データ・ストリーム・メッセージ」(86 ページ)

## srv\_negotiate

## 説明

ネゴシエーション・ログイン情報をクライアントに送信およびクライアントから受信します。

## 構文

```
CS_RETCODE srv_negotiate(spp, cmd, type)
SRV_PROC   *spp;
CS_INT     cmd;
CS_INT     type;
```

## パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*cmd*

アプリケーションがネゴシエーション・ログイン情報の送信または取得のどちらを行うために `srv_negotiate` を呼び出しているかを示します。表 3-70 に、*cmd* の有効値を示します。

**表 3-70 : *cmd* の値 (`srv_negotiate`)**

値	説明
CS_SET	<i>type</i> で定義されるネゴシエーション・ログイン情報を、クライアントに送信する。
CS_GET	<i>type</i> で定義されるネゴシエーション・ログイン情報を、クライアントから受信する。

*type*

クライアントから送信または読み取られるネゴシエーション・ログイン情報のタイプを示します。表 3-71 に、*type* の有効値を示します。

表 3-71 : type の値 (srv\_negotiate)

値	説明
SRV_NEG_CHALLENGE	ネゴシエーション・ログイン情報は、クライアントに送信されるチャレンジ・バイト・ストリーム (CS_SET)、またはクライアントから読み込まれるチャレンジ応答バイト・ストリーム (CS_GET) である。
SRV_NEG_ENCRYPT	ネゴシエーション・ログイン情報は、クライアントに送信される暗号化キーから構成される。次に、クライアントはこれを使用して、ローカル・パスワードおよびリモート・パスワードを暗号化する。このタイプは、 <i>cmd</i> が CS_SET のときにのみ有効。
SRV_NEG_EXTENDED_ENCRYPT	パスワードの暗号化に使用されるネゴシエーション・ログイン情報とパブリック・キー。これらの情報はクライアントが使用する。このタイプは、 <i>cmd</i> が CS_SET のときにのみ有効。
SRV_NEG_EXTENDED_LOCPWD	SRV_NEG_EXTENDED_ENCRYPT チャレンジに回答してクライアントから送信された暗号化されたパスワードのパブリック・キー。このタイプは、 <i>cmd</i> が CS_GET のときにのみ有効。
SRV_NEG_EXTENDED_REMPWD	ネゴシエーション・ログイン情報は、SRV_NEG_EXTENDED_ENCRYPT チャレンジに回答してクライアントから送信されたリモート・サーバ名と対応する暗号化されたパスワードのパブリック・キーが対になっている可変数。このタイプは、 <i>cmd</i> が CS_GET のときにのみ有効。
SRV_NEG_LOCPWD	SRV_NEG_ENCRYPT チャレンジに回答してクライアントから送信された暗号化されたローカル・パスワード。このタイプは、 <i>cmd</i> が CS_GET のときにのみ有効。
SRV_NEG_REMPWD	ネゴシエーション・ログイン情報は、SRV_NEG_ENCRYPT チャレンジに回答してクライアントから送信された、リモート・サーバ名と暗号化されたリモート・パスワードが対になっている可変数。このタイプは、 <i>cmd</i> が CS_GET のときにのみ有効。
SRV_NEG_SECLABEL	ネゴシエーション・ログイン情報は、クライアントに送信されるセキュリティ・ラベルまたはクライアントからサーバに送信された、セキュリティ・ラベルのセットの要求である。

値	説明
SRV_NEG_SECSSESSION	フル・パススルー・ゲートウェイ・アプリケーションがネゴシエーション・ログイン情報を使用して、ゲートウェイ・クライアントとリモート・サーバ間のダイレクト・セキュリティ・セッションを確立する。これは、チャレンジに 応答するセキュリティ・ネゴシエーションと同様である。「 <a href="#">セキュリティ・サービス</a> 」(196 ページ)を参照してください。
CS_USER_MSGID から CS_USER_MAX_MSGID の間の整数値	ネゴシエーション・ログイン情報は、 <i>type</i> 引数 自体で識別される、アプリケーション定義ハンド ドシェイクの一部。

戻り値

**表 3-72 : 戻り値 (srv\_negotiate)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype.
**
CS_RETCODE      ex_srv_negotiate PROTOTYPE((
SRV_PROC        *sproc
));

/*
** EX_SRV_NEGOTIATE
** An example routine to retrieve negotiated login information
** by using srv_negotiate.
**
** Arguments:
** sproc        A pointer to an internal thread control structure.
**
** Returns:
** CS_SUCCEED   The login information was retrieved.
** CS_FAIL      An error was detected.
**
CS_RETCODE      ex_srv_negotiate(sproc)
SRV_PROC        *sproc;
{
    /*
```

```

** Check to make sure that the thread control structure is
** not NULL.
*/
if ( sproc == (SRV_PROC *)NULL )
{
    return(CS_FAIL);
}

/* Now get the login information.*/
if ( srv_negotiate(sproc, CS_GET, SRV_NEG_CHALLENGE) ==          CS_FAIL )
{
    return(CS_FAIL);
}

return (CS_SUCCEED);
}

```

#### 使用法

- `srv_negotiate` は、ネゴシエーション・ログイン情報をクライアントに送信し、クライアントからネゴシエーション・ログイン応答を受信するために使用されます。
- `srv_negotiate` によって、**Open Server** アプリケーションはその `SRV_CONNECT` イベント・ハンドラ内でセキュア・ログイン・プロセスを実行できます。安全なコンピューティング環境においては、接続時にクライアントの身元を確認するために、アプリケーションはネゴシエーション・ログインのチャレンジと暗号化パスワードを発行することによって、より厳格に認証することもできます。
- **Open Server** は、ログインを認証するために、`SRV_CONNECT` イベント・ハンドラ内からクライアントにチャレンジまたは暗号化パスワードを送信することを選択できます。
- 一度アプリケーションがネゴシエーション・ログイン・チャレンジまたは暗号化パスワードを送信すると、接続プロセスを継続する前にクライアントの応答を読み込まなければなりません。
- **Open Server** アプリケーションは、ログイン試行を認証するために必要なだけ、チャレンジまたは応答を繰り返すことができます。ただし、アプリケーションは、次のチャレンジを送信する前に、各チャレンジの応答を読み込まなければなりません。
- 一度ネゴシエーション・ログイン・チャレンジがクライアントに送信されると、アプリケーションは、応答を読み取らなければ接続プロセスを継続することはできません。

- Open Server アプリケーションは、いかなるタイプのチャレンジであっても `srv_senddone` の呼び出しで区切らなければなりません。アプリケーションが応答を読む前にいくつかのチャレンジの「バッチ」を発行する場合は、最後のバッチを除いて、各チャレンジの後に `SRV_DONE_MORE` の `status` 引数で `srv_senddone` を呼び出さなければなりません。バッチの最後のチャレンジの後に、`SRV_DONE_FINAL` の `status` 引数で `srv_senddone` を呼び出さなければなりません。
- アプリケーション定義のハンドシェイクにおいては、ハンドシェイクの種類を設定する (`CS_SET`) ため、またはクライアントが応答する返答のタイプを指定する (`CS_GET`) ために、Open Server アプリケーションは、`type` 引数を `CS_USER_MSGID` および `CS_USER_MAX_MSGID` の間の値に設定できます。Open Server アプリケーションが予期しない値を受信した場合は、Open Server はエラーを発行します。
- クライアントがチャレンジまたは暗号化パスワードに応答する場合は、`srv_negotiate` はクライアントの応答を受信するまでスレッドの実行を中断します。アプリケーションについては、セキュア `SRV_CONNECT` イベント・ハンドラをコーディングする場合は、このことを念頭に置いてください。
- ネゴシエーション・ログイン・チャレンジと応答は、`srv_bind`、`srv_descfmt`、`srv_xferdata` によって送受信されるパラメータを介してデータ値を運びます。この3つのルーチンは、ネゴシエーション・ログイン・データに対する定義またはアクセスを行うために `SRV_NEGDATA` の `type` 引数を受け取ります。
- [表 3-73](#) に、クライアントに送られる各チャレンジに伴うパラメータを示します。



表 3-73 : 必要なチャレンジ・パラメータ (srv\_negotiate)

ネゴシエーション・ログイン・タイプ	必要なパラメータ
SRV_NEG_CHALLENGE	パラメータは1つで、チャレンジ・データ値。データ型は、CS_DATAFMT <i>status</i> フィールドをCS_CANBENULL に設定したCS_BINARY_TYPE。
SRV_NEG_ENCRYPT	パラメータは1つで、暗号化キー・データ値。データ型は、CS_DATAFMT <i>status</i> フィールドをCS_CANBENULL に設定したCS_BINARY_TYPE。
SRV_NEG_SECLABEL	パラメータなし。
SRV_NEG_SECSESSION	セキュリティ・セッション・コールバックはパラメータの数とそのデータ・フォーマットを指定。詳細については、「 <a href="#">セキュリティ・セッション・コールバック</a> 」(222 ページ)および『Open Client Library/C リファレンス・マニュアル』を参照。
CS_USER_MSGID から CS_USER_MAX_MSGID の間の整数値	パラメータは1つで、アプリケーション定義のログイン・ハンドシェイク・データ値。

- 表 3-74 に、ネゴシエーション・ログイン・チャレンジのタイプ別にクライアントから読み込む必要のあるパラメータを示します。

**表 3-74 : 予期されるチャレンジ・パラメータ (srv\_negotiate)**

ネゴシエーション・ログイン・タイプ	存在するパラメータ
SRV_NEG_CHALLENGE	パラメータは1つで、チャレンジ応答データ。
SRV_NEG_LOCPWD	パラメータは1つで、暗号化ローカル・パスワード。
SRV_NEG_REMPWD	サーバ名/パスワードが対になっている可変数。
SRV_NEG_SECLABEL	次の4つのパラメータ。 Param 1 : 最大読み込みレベル・ラベル。 Param 2 : 最大書き込みレベル・ラベル。 Param 3 : 最小書き込みレベル・ラベル。 Param 4 : 現在の書き込みレベル・ラベル。
SRV_NEG_SECSSESSION	セキュリティ・セッション・コールバックはパラメータの数とそのデータ・フォーマットを指定。詳細については、「 <a href="#">セキュリティ・セッション・コールバック</a> 」(222ページ)および『Open Client Library/C リファレンス・マニュアル』を参照。
CS_USER_MSGID から CS_USER_MAX_MSGID の間の整数値	パラメータは1つで、アプリケーション定義のログイン・ハンドシェイク・データ値。

- パスワード暗号化チャレンジに対する応答である SRV\_NEG\_ENCRYPT は、パラメータの2つのセットから構成されることもあることに注意してください。SRV\_NEG\_LOCPWD 応答は、クライアントの暗号化パスワードを示すパラメータを渡します。また、クライアントは、クライアントの暗号化リモート・サーバ・パスワードとリモート・サーバ名をそれぞれ示すパラメータを渡す SRV\_NEG\_REMPWD 応答を送信することもできます。SRV\_NEG\_ENCRYPT チャレンジに対する SRV\_NEG\_LOCPWD 応答は、常に存在します。クライアントによってリモート・サーバ・パスワードがまったく送られていない場合には、SRV\_NEG\_REMPWD 応答を受ける要求は失敗します。

- **Open Client** と **Open Server** を使用してゲートウェイの機能を実装するアプリケーションは、クライアントとリモート・サーバ間においてネゴシエーション・ログイン・チャレンジと応答を転送するために、**Open Client** のネゴシエーション・ログイン・コールバック・メカニズムを使用しなければなりません。このようなアプリケーションでは、**Open Client** ネゴシエーション・ログイン・コールバックは、クライアントにチャレンジを転送してその応答を受信する（これを **Open Client** はリモート・サーバに返します）ために必要な **Server-Library** ルーチン呼び出しを含んでいなければなりません。

ゲートウェイ・アプリケーションがクライアントとリモート・サーバ間のダイレクト・セキュリティ・セッションを確立する場合、**Open Client** セキュリティ・セッション・コールバックが必要です。このコールバックには、内部が隠されたセキュリティ・トークンをクライアントに転送してその応答を受信するために必要な **Server-Library** 呼び出しが含まれている必要があります。その後、この応答は **Open Client** によってリモート・サーバに返されません。詳細については、「[セキュリティ・セッション・コールバック](#)」(222 ページ) および『[Open Client Client-Library/C リファレンス・マニュアル](#)』を参照してください。

参照 [srv\\_senddone, srv\\_thread\\_props](#)

## srv\_numparams

**説明** 現在のクライアント・コマンドに含まれているパラメータの数を返します。

**構文** `CS_RETCODE srv_numparams(spp, numparamsp)`

```
SRV_PROC   *spp;
CS_INT     *numparamsp;
```

**パラメータ** *spp*  
内部スレッド制御構造体へのポインタです。

*numparamsp*  
現在のクライアント・コマンドまたはカーソル・データ・ストリームの引数の数へのポインタは、\**numparamsp* で返されます。

戻り値

**表 3-75 : 戻り値 (srv\_numparams)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_numparams PROTOTYPE((
SRV_PROC        *spp,
CS_INT          *countp
));

/*
** EX_SRV_NUMPARAMS
**
** Example routine to illustrate the use of srv_numparams to
** get the number parameters contained in the current client
** command.
**
** Arguments:
** spp      A pointer to an internal thread control structure.
** countp  A pointer to the buffer in which the number of
**          parameters in the client command is returned.
**
** Returns:
**
** CS_SUCCEED  The number of parameters was successfully
** returned.
** CS_FAIL    An error was detected.
*/
CS_RETCODE      ex_srv_numparams(spp, countp)
SRV_PROC        *spp;
CS_INT          *countp;
{
    if (srv_numparams(spp, countp) == CS_FAIL)
        return(CS_FAIL);

    return (CS_SUCCEED);
}
```

## 使用法

- `srv_numparams` は、現在の MSG、RPC、DYNAMIC、カーソル・データ・ストリームのパラメータの数、または `srv_negotiate(CS_GET)` 呼び出しに対するクライアントの応答に含まれるパラメータの数を返します。この数は、ランタイム時に Open Server が満たすデフォルト・パラメータを含みます。
- `srv_numparams` は、特定のイベントのためのハンドラからしか呼び出すことができません。表 3-76 に、それらのイベントとそのパラメータを示します。

表 3-76 : イベントとパラメータ (`srv_numparams`)

イベント	パラメータ
SRV_CURSOR	カーソル・パラメータ
SRV_RPC	RPC パラメータ
SRV_DYNAMIC	動的 SQL パラメータ
SRV_MSG	MSG パラメータ
SRV_LANGUAGE	言語パラメータ。言語ハンドラ内でパラメータ・データのチェックと取得を行うため、 <code>srv_numparams</code> は 5.0 またはそれ以上の TDS レベルが要求される。接続の TDS レベルをチェックするコードをアプリケーションに追加し、その TDS バージョンが <code>SRV_TDS_5_0</code> 未満であれば、 <code>srv_numparams</code> を省略する必要がある。 <code>srv_props</code> ルーチンの <code>SRV_S_TDSVERSION</code> プロパティを使用すると、接続の TDS プロトコル・バージョンを取得できる (表 2-25 (160 ページ) を参照)。
<code>srv_negotiate (CS_GET)</code> 呼び出しの後	クライアントの応答内のパラメータ。たとえば、サンプル・プログラム <code>ctos.c</code> 内など。

## 参照

[srv\\_bind](#), [srv\\_cursor\\_props](#), [srv\\_descfmt](#), [srv\\_dynamic](#), [srv\\_msg](#), [srv\\_xferdata](#), 「パラメータとロー・データの処理」 (152 ページ)

## srv\_options

## 説明

オプション情報をクライアントに送信、またはクライアントから受信する。

## 構文

```
CS_RETURNCODE srv_options(spp, cmd, optcmdp, optionp,
                          bufp, bufsize, outlenp)
```

```

SRV_PROC  *spp;
CS_INT    cmd;
CS_INT    *optcmdp;
CS_INT    *optionp;
CS_CHAR   *bufp;
CS_INT    bufsize;
CS_INT    *outlenp;
    
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

アプリケーションがオプション情報の送信または受信のどちらを行うために *srv\_options* を呼び出しているかを示します。表 3-77 に、*cmd* の有効値を示します。

**表 3-77 : *cmd* の値 (*srv\_options*)**

値	説明
CS_SET	Open Server アプリケーションはクライアントにオプション・コマンドを送信している。
CS_GET	Open Server アプリケーションはクライアントからオプション・コマンドを受信している。

*optcmdp*

クライアントのオプション・コマンド (CS\_GET) を含むプログラム変数、または Open Server アプリケーションのオプション・コマンド (CS\_SET) を含むプログラム変数のいずれかへのポインタです。表 3-78 に、\**optcmdp* の有効値を示します。

表 3-78 : optcmdp の値 (srv\_options)

値	説明	Cmd
SRV_SETOPTION	クライアントはオプションの設定を要求している。 <i>optionp</i> と関連される値は、 <i>*bufp</i> で返される。 Open Server は、返されるデータ・サイズのバイト数を <i>bufsize</i> に設定する。 <i>*bufp</i> がすべてのデータを保持できない場合は、この関数は CS_FAIL を返し、オプション値の実際のサイズは <i>*outlenp</i> でバイト数を返し、 <i>optionp</i> と <i>bufp</i> の値は未定義になる。	CS_GET
SRV_CLEAROPTION	クライアントは、 <i>optionp</i> がデフォルト値に設定されることを要求している。 <i>bufp</i> と <i>optionp</i> の値は未定義のままとなる。	CS_GET
SRV_GETOPTION	クライアントは、 <i>*optionp</i> の現在の値に関する情報を要求している。 <i>bufp</i> と <i>optionp</i> の値は未定義のままとなる。	CS_GET
SRV_SENDOPTION	アプリケーションは、 SRV_GETOPTION コマンドに回答して現在のオプション値をクライアントに送っている。 <i>bufp</i> はこのオプションに関連する引数を指し、 <i>bufsize</i> は <i>*bufp</i> にあるデータ・サイズのバイト数を持っている。	CS_SET

*optionp*

クライアントが要求したオプション (CS\_GET) または Open Server アプリケーションが応答に使用しているオプション (CS\_SET) のいずれかを指すポインタです。

*bufp*

オプション (CS\_GET) に関連する値または要求者に送信されるオプション (CS\_SET) の値のいずれかを持つバッファを指すポインタです。 *\*optionp* は問題のオプションを含み、 *\*bufp* はその値を含んでいます (CS\_SET において)。オプションとその有効値の完全なリストについては、次の表を参照してください。

*bufsize*

*\*bufp* バッファの長さです。文字列オプション値を持つオプションを送信するときに、 *bufp* が null で終了している場合は、 *bufsize* を CS\_NULLTERM として転送してください。

*outlenp*

*\*buff* で返されているオプション値のサイズのバイト数に設定されたプログラム変数へのポインタです。このパラメータはオプションで、*cmd* が CS\_GET に設定された場合にのみ使用されます。

戻り値

**表 3-79 : 戻り値 (srv\_options)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype
*/
CS_RETCODE      ex_srv_options PROTOTYPE((
SRV_PROC        *spp,
CS_INT          *rowcount
));

/*
** EX_SRV_OPTIONS
**
** Example routine to recieve option information for the
** maximum number of regular rows to return (CS_OPT_ROWCOUNT)
** from a client.
**
** Arguments:
** spp          A pointer to an internal thread control structure.
** rowcount    Return pointer for the number of rows to return.
**
** Returns:
**
** CS_SUCCEED  Successfully retrieved option.
** CS_FAIL     An error was detected.
*/
CS_RETCODE      ex_srv_options(spp, rowcount)
SRV_PROC        *spp;
CS_INT          *rowcount;
{
    CS_INT      optcmdp;    /* The client's option command.*/
    CS_INT      optionp;   /* The client's option request.*/

    /* Initialization.*/
```



```
optcmdp = SRV_GETOPTION;
optionp = CS_OPT_ROWCOUNT;

/*
** Get the maximum number of rows to return.
*/
if (srv_options(spp, CS_GET, &optcmdp, &optionp, (CS_VOID
    *)rowcount, CS_SIZEOF(CS_INT), (CS_INT *)NULL) !=
    CS_SUCCEED)
{
    return(CS_FAIL);
}
return (CS_SUCCEED);
}
```

**使用法**

- `srv_options` は Open Server アプリケーションがクライアントからオプション情報を読み取ったり、クライアントにオプション情報を送信したりすることを可能にします。
- [表 3-80](#) に、有効なオプション、その有効値、および `optionp` パラメータのデータ型を示します。

**表 3-80 : オプションの説明 (srv\_options)**

オプション	有効値	bufp が指す対象
CS_OPT_ANSINULL	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_ANSIPERM	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_ARITHABORT	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_ARITHIGNORE	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_AUTHOFF	CS_OPT_SA、 CS_OPT_SSO、 CS_OPT_OPER	文字列
CS_OPT_AUTHON	CS_OPT_SA、 CS_OPT_SSO、 CS_OPT_OPER	文字列
CS_OPT_CHAINXACTS	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_CURCLOSEONXACT	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_CURREAD	読み込みラベル ( 文字列 )	文字列
CS_OPT_CURWRITE	書き込みラベル ( 文字列 )	文字列
CS_OPT_DATEFIRST	CS_OPT_SUNDAY CS_OPT_MONDAY CS_OPT_TUESDAY CS_OPT_WEDNESDAY CS_OPT_THURSDAY CS_OPT_FRIDAY CS_OPT_SATURDAY	週の最初の日に使用する曜日を 示す記号値
CS_OPT_DATEFORMAT	CS_OPT_FMTMDY CS_OPT_FMTDMY CS_OPT_FMTYMD CS_OPT_FMTYDM CS_OPT_FMTMYD CS_OPT_FMTDYM	日時値で使用する年、月、日の順序を表す記号値
CS_OPT_FIPSFLAG	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_FORCEPLAN	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_FORMATONLY	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_GETDATA	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_IDENTITYOFF	テーブル名を表す文字列値	文字列
CS_OPT_IDENTITYON	テーブル名を表す文字列値	文字列
CS_OPT_ISOLATION	CS_OPT_LEVEL1 CS_OPT_LEVEL3	分離レベルを表す記号値

オプション	有効値	bufp が指す対象
CS_OPT_NOCOUNT	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_NOEXEC	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_PARSEONLY	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_QUOTED_IDENT	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_RESTREES	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_ROWCOUNT	返す通常ローの最大数	CS_INT 0 が指定されるとすべてのローが返される。
CS_OPT_SHOWPLAN	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_STATS_IO	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_STATS_TIME	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_STR_RTRUNC	CS_TRUE、CS_FALSE	CS_BOOL
CS_OPT_TEXTSIZE	サーバが返す最長の text または image の値	CS_INT
CS_OPT_TRUNCIGNORE	CS_TRUE、CS_FALSE	CS_BOOL

「オプション」(138 ページ)には、各オプションの説明とそれぞれのデフォルト値を示しています。

- Open Server はクライアントから受信した各オプション・コマンドに対して、SRV\_OPTION イベントを発生させます。SRV\_OPTION イベント・ハンドラ内で、アプリケーションはオプション情報を取得するために *cmd* を CS\_GET に設定して *srv\_options* を呼び出すことができます。*srv\_options* が戻る場合、*optcmdp*、*optionp*、\**bufp* にはクライアントから受け取ったオプション情報がすべて格納されています。SRV\_OPTION イベント・ハンドラ以外のイベント・ハンドラから *srv\_options* を呼び出すと、エラーになります。
- SRV\_SETOPTION と SRV\_CLEAROPTION への応答では、アプリケーションは引数を SRV\_DONE\_FINAL に指定した *srv\_senddone* を呼び出さなければなりません。オプション処理が失敗した場合には、アプリケーションは、引数を SRV\_DONE\_FINAL | SRV\_DONE\_ERROR に指定した *srv\_senddone* を呼び出さなければなりません。
- アプリケーションは、*optcmdp* を SRV\_SENDOPTION に設定し、*bufp* がオプションの現在の値を指している状態で *srv\_options* を呼び出して受け取ったすべての SRV\_GETOPTION コマンドに応答する必要があります。

- アプリケーションは、クライアントが `SRV_SETOPTION` コマンドを使用して送信した引数を受け取るために十分なサイズの `*bufp` バッファを確保する必要があります。バッファが十分な大きさではない場合、`srv_options` は `CS_FAIL` を返し、`outlenp` は必要なサイズに設定されます。
- `Open Server` には、特定のオプションの意味に関する概念がありません。`Open Server` アプリケーションは、クライアントのオプション・コマンドを保存し、コマンドが要求するアクションを実行します。`SRV_OPTION` イベント・ハンドラがインストールされていない場合は、クライアントから受けたオプション・コマンドはエラーとなり、拒否されます。

参照

[srv\\_senddone](#), 「オプション」 (138 ページ)

## srv\_orderby

説明

クライアントに `order-by` リストを返します。

構文

```
CS_RETCODE srv_orderby(spp, numcols, collistp)
```

```
SRV_PROC   *spp;
CS_INT     numcols;
CS_INT     *collistp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*numcols*

`order-by` リストのカラムの数です。カラムは `CS_INT` の配列として渡されるので、*numcols* は、実際には *collistp* 配列の要素の数です。

*collistp*

カラム番号の配列へのポインタです。この配列のサイズは、*numcols* です。

戻り値

**表 3-81 : 戻り値 (srv\_orderby)**

戻り値	意味
<code>CS_SUCCEEDED</code>	ルーチンが正常に終了した。
<code>CS_FAIL</code>	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
```

```
** Local Prototype
*/
CS_RETCODE   ex_srv_orderby PROTOTYPE((
SRV_PROC     *spp
));

/*
** EX_SRV_ORDERBY
**
** Example routine using srv_orderby to define and return to a
** client application the order-by list for a simple SQL
** command.
** This example uses the SQL command:
**
**      "select a,b,c,d from my_tab
**       order by c,a"
**
** Arguments:
** spp    A pointer to the internal thread control structure.
**
** Returns:
** CS_SUCCEEDED    Order-by list was successfully defined.
** CS_FAIL         An error was detected..
*/
CS_RETCODE   ex_srv_orderby(spp)
SRV_PROC     *spp;
{
    /* There are two columns specified in the order-by clause.*/
    CS_INT     collist[2];
    CS_INT     numcols;

    /* Initialization.*/
    numcols = 2;

    /*
    ** Initialize the collist array in the order the
    ** columns occur in the order-by clause.
    **
    ** "c" is the 1st column specified in the order-by,
    ** and is the 3rd column specified in the select-list.
    */
    collist[0] = (CS_INT)3;
    /*
    ** "a" is the 2nd column specified in the order-by,
    ** and is the 1st column specified in the select-list.
    */
    */
}
```

```

collist[1] = (CS_INT)1;
/*
** Define the order-by list.
*/
if (srv_orderby(spp, numcols, collist) != CS_SUCCEED)
{
    return(CS_FAIL);
}
return (CS_SUCCEED);
}

```

使用法

- `srv_orderby` は、`order-by` 情報を返すという Adaptive Server Enterprise の機能を模擬的に実行する場合のみ必要です。
- `srv_orderby` では、Open Server アプリケーションがソート順に関する情報をクライアントに返すことができます。SQL コマンドは、次のようになります。

```

select a, b, c, d
order by c, a

```

ソート順は、先にカラム `c`、次にカラム `a` です。アプリケーションは、カラム番号配列において、まずカラム 3、次にカラム 1 をリストすることによって、この情報をクライアントに返します。

- `select` リストの先頭のカラムは、カラム 1 です。
- `srv_orderby` は、`srv_descfmt` の呼び出しの後、`srv_bind` の呼び出しの前に呼び出さなければなりません。

## srv\_poll (UNIX のみ)

説明

一連のオープン・ストリームのファイル記述子の I/O イベントをチェックします。

構文

```

CS_INT srv_poll(fdsp, nfds, waitflag)
SRV_POLLFD *fdsp;
CS_INT nfds;
CS_INT waitflag;

```

パラメータ

*fdsp*

対象の各オープン・ファイル記述子に対して要素を 1 つずつ持っている `SRV_POLLFD` 構造体の配列を指すポインタです。`SRV_POLLFD` 構造体には、次のメンバがあります。

```

CS_INT   srv_fd;           /* File descriptor.*/
CS_INT   srv_events;      /* Relevant events.*/
CS_INT   srv_revents;     /* Returned events.*/

```

***nfds***

*\*fdsp* 配列の要素の数です。

***waitflag***

目的のオペレーションのためのファイル記述子を使用可能になるまで、スレッドが休止されるのかどうかを示す **CS\_INT** 値です。

**SRV\_M\_WAIT** に設定すると、スレッドは休止し、指定されたオペレーションに対して *\*fdsp* 配列のファイル記述子 (いずれでも) が使用可能になると、ウェイクアップします。フラグが

**SRV\_M\_NOWAIT** に設定されていると、**srv\_poll** はチェックを行い、呼び出し元に戻ります。リターン・ステータスの値がゼロより大きい場合は、目的のオペレーションのファイル記述子を使用可能であったことを示します。

## 戻り値

**表 3-82 : 戻り値 (srv\_poll)**

戻り値	意味
整数	使用可能なファイル記述子の数。
-1	ルーチンが失敗。
0	使用可能なファイル記述子がない。

## 例

```

#include          <ospublic.h>
/*
** Local Prototype
*/
CS_RETCODE      ex_srv_pollPROTOTYPE((
struct pollfd   *fdp,
CS_INT          nfds
));

/*
** EX_SRV_POLL
**
** This routine demonstrates how to use srv_poll to poll
** application-specific file descriptors.
**
** Arguments:
**     fdp - The address of the file descriptor array.
**     nfd - The number of file descriptors to poll.
**
** Returns

```

```

**
**      CS_SUCCEEDED      If the data address is returned.
**      CS_FAIL           If the call to srv_poll failed.
**
*/
CS_RETCODE      ex_srv_poll(fdp, nfds)
struct pollfd   *fdp;
CS_INT          nfds;
{
    /*
    ** Initialization.
    */
    lp = (CS_VOID *)NULL;

    /*
    ** Calls srv_poll to check if any of these file
    ** descriptors are active; ask to sleep until at
    ** least one of them is.
    */
    if( srv_poll(fdp, nfds, SRV_M_WAIT) == (CS_INT)-1 )
    {
        return CS_FAIL;
    }

    /*
    ** All done.
    */
    return CS_SUCCEEDED;
}

```

使用方法

- アプリケーションはファイル記述子を調べるため、または実行する I/O があるまでスレッドを休止するために、`srv_poll` を使用することができます。
- [表 3-83](#) に、`srv_events` と `srv_revents` の有効値を示します。



表 3-83 : `srv_events` および `revents` の値 (`srv_poll`)

値	説明
<code>SRV_POLLIN</code>	通常の読み込みイベント。
<code>SRV_POLLPRI</code>	優先イベントを受け取った。
<code>SRV_POLLOUT</code>	ファイル記述子は書き込みが可能。
<code>SRV_POLLERR</code>	ファイル記述子でエラーが起こった。
<code>SRV_POLLHUP</code>	ファイル記述子でハングした。この値は、リターン・イベントでのみ有効。
<code>SRV_POLLNVAL</code>	<code>SRV_POLLFD</code> で無効なファイル記述子が指定された。

- `srv_poll` は、すべての UNIX プラットフォームで使用できます。

**注意** ネイティブの `poll(2)` システム・コールをサポートする UNIX プラットフォームでアプリケーションが `srv_poll` を使用する場合、アプリケーションは `ospublic.h` の前に `<sys/poll.h>` を含める必要があります。

参照

[srv\\_capability](#), [srv\\_select \(UNIX のみ\)](#)

## srv\_props

説明

Open Server プロパティを定義または取得します。

構文

```
CS_RETCODE srv_props(cp, cmd, property, bufp, buflen,
                    outlenp)
```

```
CS_CONTEXT *cp;
CS_INT cmd;
CS_INT property;
CS_VOID *bufp;
CS_INT buflen;
CS_INT *outlenp;
```

パラメータ

*scp*

`cs_ctx_alloc` を使用して先に割り付けた `CS_CONTEXT` 構造体へのポインタです。

*cmd*

実行するアクションです。表 3-84 に、*cmd* の有効値を示します。

**表 3-84 : cmd の値 (srv\_props)**

値	意味
CS_SET	Open Server アプリケーションがプロパティを設定している。この場合、 <i>bufp</i> は、プロパティが設定される値を持ち、 <i>buflen</i> は、その値のサイズのバイト数でなければならない。
CS_GET	Open Server アプリケーションがプロパティを取得している。この場合、 <i>bufp</i> は、プロパティ値が置かれているバッファを指し、 <i>buflen</i> は、そのバッファのサイズのバイト数でなければならない。
CS_CLEAR	Open Server アプリケーションは、プロパティをデフォルト値にリセットしている。この場合は、 <i>bufp</i> 、 <i>buflen</i> 、 <i>outlenp</i> は無視される。

*property*

設定、取得、またはクリアされるプロパティです。この引数の有効値のリストについては、「使用法」にある表を参照してください。

*bufp*

プロパティ値の情報が設定される (CS\_SET)、またはプロパティ値の情報が取得される (CS\_GET) Open Server アプリケーションのデータ・バッファへのポインタです。

*buflen*

バッファ長をバイト数で示したものです。

*outlenp*

取得されたプロパティ値の長さのバイト数を設定するために Open Server が使用する CS\_INT 変数へのポインタです。この引数はオプションであり、*cmd* が CS\_GET のときのみ使用します。

戻り値

**表 3-85 : 戻り値 (srv\_props)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local prototype
*/
CS_RETCODE ex_srv_set_propPROTOTYPE((
CS_CONTEXT *cp,
CS_INT property,
CS_VOID *bufp,
CS_INT buflen
```

```

));
/*
** EX_SRV_SET_PROP
**
** Example routine to set a property using srv_props.
**
** Arguments:
**
** *cp      Pointer to a CS_CONTEXT structure previously
**          allocated by cs_ctx_alloc.
** property The property being set.
** *bufp    Pointer to the value the property is to be
**          set to.
** buflen   The length of the value.
**
** Returns
**
** CS_SUCCEEDED Arguments were valid and srv_props was called.
** CS_FAIL       An error was detected.
**
*/
CS_RETCODE      ex_srv_set_prop(cp, property, bufp, buflen)
CS_CONTEXT      *cp;
CS_INT          property;
CS_VOID         *bufp;
CS_INT          buflen;
{
    /* Check arguments.*/
    if(cp == (CS_CONTEXT *)NULL)
    {
        return(CS_FAIL);
    }
    if(buflen < 1)
        return(CS_FAIL);
    return(srv_props(cp, (CS_INT)CS_SET, property, bufp, buflen,
        (CS_INT *)0));
}

```

#### 使用法

- サーバワイドな設定パラメータとプロパティを定義および取得するために、`srv_props` が呼び出されます。
- `srv_version` を呼び出してから、`srv_props` を呼び出します。
- `srv_init` を呼び出す前に、`SRV_S_TRACEFLAG`、`SRV_S_LOGFILE`、および `SRV_S_TRUNCATELOG` を除くすべてのプロパティが `srv_props` によって設定されていなければなりません。

- `srv_init` を呼び出した後、`bufp` を空の文字列 ("") に設定し、`buflen` をゼロに設定した状態で `SRV_S_LOGFILE` プロパティを設定すると、ログ・ファイルが閉じます。
- TCP ソケットの入出力バッファ・サイズを設定するためのサーバ・プロパティは、`SRV_S_TCP_RCVBUF` と `SRV_S_TCP_SNDBUF` です。Open Server アプリケーションは、これらのプロパティ設定を使用して、オペレーティング・システムの `setsockopt` コマンドでバッファ・サイズを設定します。`setsockopt` は TCP の `connect` コマンドおよび `accept` コマンドの前に呼び出す必要があるため、これらのプロパティを設定してから接続の確立を試みてください。

これらのプロパティをアプリケーションに応じて設定します。たとえば、クライアントが大量のデータをサーバに送信することが想定される場合は、`SRV_S_TCP_RCVBUF` を大きな値に設定して対応するバッファ・サイズを増加します。

---

**注意** `CS_TCP_RCVBUF` および `CS_TCP_SNDBUF` の各 context/connection プロパティを使用して、クライアント側で TCP ソケットの入出力バッファ・サイズを設定します。『Open Client Library/C リファレンス・マニュアル』を参照してください。

---

サーバの例：

```
srv_props(cp, CS_SET, SRV_S_TCP_SNDBUF, bufp,
CS_SIZEOF(CS_INT), (CS_INT *)NULL);
```

- 表 3-86 に、サーバ・プロパティ値、設定や取得の可否、それぞれのプロパティ値のデータ型を示します。

**表 3-86 : サーバのプロパティとそのデータ型 (srv\_props)**

プロパティ	設定／クリア	取得	cmd が CS_SET の場合の bufp	cmd が CS_GET の場合の bufp
SRV_S_ALLOCFUNC	あり	あり	関数ポインタ	関数ポインタのアドレス
SRV_S_APICHK	あり	あり	CS_BOOL	CS_BOOL
SRV_S_ATTREASON	なし	あり	適用しない	CS_INT
SRV_S_CERT_AUTH	あり	あり	char*	char*
SRV_S_CURTHREAD	なし	あり	適用しない	スレッド・ポインタのアドレス
SRV_S_DISCONNECT	あり	あり	CS_BOOL	CS_BOOL
SRV_S_DEFQUEUESIZE	あり	あり	CS_INT	CS_INT

プロパティ	設定/ クリア	取得	cmd が CS_SET の場 合の bufp	cmd が CS_GET の場合 の bufp
SRV_S_DS_PROVIDER	あり	あり	文字列へのポ インタ	文字列へのポインタ
SRV_S_DS_REGISTER	あり	あり	CS_BOOL	CS_BOOL
SRV_S_ERRHANDLE	あり	あり	関数ポインタ	関数ポインタのアドレス
SRV_S_FREEFUNC	あり	あり	関数ポインタ	関数ポインタのアドレス
SRV_S_IFILE	あり	あり	文字列	文字列
SRV_S_LOGFILE	あり	あり	文字列	文字列
SRV_S_LOGSIZE	あり	あり	CS_INT	CS_INT
SRV_S_MAXLISTENERS	あり	あり	CS_INT	CS_INT
SRV_S_MSGPOOL	あり	あり	CS_INT	CS_INT
SRV_S_NETBUFSIZE	あり	あり	CS_INT	CS_INT
SRV_S_NETTRACEFILE	あり	あり	文字列	文字列
SRV_S_NUMCONNECTIONS	あり	あり	CS_INT	CS_INT
SRV_S_NUMLISTENERS	なし	あり	CS_INT	CS_INT
SRV_S_NUMMSGQUEUES	あり	あり	CS_INT	CS_INT
SRV_S_NUMMUTEXES	あり	あり	CS_INT	CS_INT
SRV_S_NUMREMBUF	あり	あり	CS_INT	CS_INT
SRV_S_NUMREMSITES	あり	あり	CS_INT	CS_INT
SRV_S_NUMTHREADS	あり	あり	CS_INT	CS_INT
SRV_S_NUMUSEREVENTS	あり	あり	CS_INT	CS_INT
SRV_S_PREEMPT	あり	あり	CS_BOOL	CS_BOOL
SRV_S_REALLOCFUNC	あり	あり	関数ポインタ	関数ポインタのアドレス
SRV_S_RETPARMS	あり	あり	CS_BOOL	CS_BOOL
SRV_S_REQUESTCAP	あり	あり	CS_CAP_TYP E 構造体	CS_CAP_TYPE 構造体
SRV_S_RESPONSECAP	あり	あり	CS_CAP_TYP E 構造体	CS_CAP_TYPE 構造体
SRV_S_SEC_KEYTAB	あり	あり	文字列へのポ インタ	文字列へのポインタ
SRV_S_SEC_PRINCIPAL	あり	あり	文字列へのポ インタ	文字列へのポインタ
SRV_S_SERVERNAME	なし	あり	文字列	文字列
SRV_S_SSL_CIPHER	あり	なし	char*	
SRV_S_SSL_LOCAL_ID	あり	あり	struct	char*
SRV_S_SSL_VERSION	あり	なし	CS_INT	
SRV_S_STACKSIZE	あり	あり	CS_INT	CS_INT

プロパティ	設定／ クリア	取得	cmd が CS_SET の場 合の bufp	cmd が CS_GET の場合 の bufp
SRV_S_TCP_RCVBUF	あり	あり	CS_INT	CS_INT
SRV_S_TCP_SNDBUF	あり	あり	CS_INT	CS_INT
SRV_S_TDSVERSION	あり	あり	CS_INT	CS_INT
SRV_S_TIMESLICE	あり	あり	CS_INT	CS_INT
SRV_S_TRACEFLAG	あり	あり	CS_INT ( ビッ トマスク )	CS_INT ( ビットマスク )
SRV_S_TRUNCATELOG	あり	あり	CS_BOOL	CS_BOOL
SRV_S_USESRVLANG	あり	あり	CS_BOOL	CS_BOOL
SRV_S_VERSION	なし	あり	適用しない	文字列
SRV_S_VIRTCLKRATE	あり	あり	CS_INT	CS_INT
SRV_S_VIRTTIMER	あり	あり	CS_BOOL	CS_BOOL

- 表 3-87 に、各サーバ・プロパティのデフォルト値を示します。

表 3-87 : プロパティの有効値とそのデフォルト値 (srv\_props)

プロパティ	デフォルト
SRV_S_ALLOCFUNC	malloc()
SRV_S_APICLK	CS_TRUE
SRV_S_ATTNREASON	デフォルトなし
SRV_S_CURTHREAD	適用せず
SRV_S_DEFQUEUESIZE	SRV_DEF_DEFQUEUESIZE
SRV_S_DISCONNECT	CS_FALSE
SRV_S_DS_PROVIDER	プラットフォームに依存する。詳細については、使用しているプラットフォームの『Open Client/Server 設定ガイド』を参照。
SRV_S_DS_REGISTER	CS_TRUE、Server-Library が起動時に自身でディレクトリに登録する
SRV_S_ERRHANDLE	エラー・ハンドラなし
SRV_S_FREEFUNC	free()
SRV_S_IFILE	<i>\$\$SYBASE/interfaces</i>
SRV_S_LOGFILE	<i>srv.log</i>
SRV_S_LOGSIZE	最大整数値
SRV_S_MAXLISTENERS	CS_MAX_NOMAX
SRV_S_MSGPOOL	SRV_DEF_MSGPOOL
SRV_S_NETBUFSIZE	SRV_DEF_NETBUFSIZE
SRV_S_NETTRACEFILE	<i>sybnet.dbg</i>
SRV_S_NUMCONNECTIONS	SRV_DEF_NUMCONNECTIONS
SRV_S_NUMLISTENERS	適用せず
SRV_S_NUMMSGQUEUES	SRV_DEF_NUMMSGQUEUES
SRV_S_NUMMUTEXES	SRV_DEF_NUMMUTEXES
SRV_S_NUMREMBUF	SRV_DEF_NUMREMBUF
SRV_S_NUMREMSITES	SRV_DEF_NUMREMSITES
SRV_S_NUMTHREADS	SRV_DEF_NUMTHREADS
SRV_S_NUMUSEREVENTS	SRV_DEF_NUMUSEREVENTS
SRV_S_PREEMPT	CS_FALSE
SRV_S_REALLOCFUNC	realloc()
SRV_S_REQUESTCAP	「機能」(25 ページ)を参照してください。
SRV_S_RESPONSECAP	「機能」(25 ページ)を参照してください。
SRV_S_RETPARMS	デフォルトなし
SRV_S_SEC_KEYTAB	デフォルトなし
SRV_S_SEC_PRINCIPAL	セキュリティ・メカニズムに依存する。
SRV_S_SERVERNAME	DSLISTEN 環境変数

プロパティ	デフォルト
SRV_S_STACKSIZE	SRV_DEF_STACKSIZE
SRV_S_TCP_RCVBUF	プラットフォームに依存する。
SRV_S_TCP_SNDBUF	プラットフォームに依存する。
SRV_S_TDSVERSION	SRV_TDS_5_0
SRV_S_TIMESLICE	SRV_DEF_TIMESLICE
SRV_S_TRACEFLAG	0
SRV_S_TRUNCATELOG	CS_FALSE
SRV_S_USESRVLANG	CS_TRUE
SRV_S_VERSION	コンパイル時のバージョン文字列
SRV_S_VIRTCLKRATE	SRV_DEF_VIRTCLKRATE
SRV_S_VIRTTIMER	CS_FALSE

- デフォルト値が割り当てられていて設定可能なすべてのサーバ・プロパティは、*cmd* を **CS\_CLEAR** に設定した状態で *srv\_props* を呼び出すことによって、デフォルト値にリセットできます。
- すべてのサーバ・プロパティは、*cmd* を **CS\_GET** に設定して *srv\_props* を呼び出すことによって、いつでも取得できます。Open Server アプリケーションがプロパティの値を定義していない場合は、デフォルト値が返されます。
- プロパティについては、「プロパティ」のページを参照してください。
- プロパティが取得されているときに、ユーザ・バッファのサイズがプロパティ値を格納するには十分なサイズではないことを *buflen* が示した場合には、Open Server は必要なバイト数を *\*outlenp* に設定し、ユーザ・バッファは変更されません。
- デフォルトの *stacksize* (**SRV\_S\_STACKSIZE** のデフォルト値) は、使用するプラットフォームに応じて異なります。

Open Server のネイティブスレッド・バージョンでは、基本となるスレッドのデフォルトの *stacksize* が使用されます。この値は、**SRV\_S\_STACKSIZE** プロパティで *stacksize* を設定することにより変更できます。

*stacksize* を設定する場合、その *stacksize* が小さすぎるときは、スタック・オーバーフロー・エラーが発生する可能性があることに注意してください。

参照

[srv\\_init](#), [srv\\_thread\\_spawn](#), [srv\\_spawn](#), 「プロパティ」 (157 ページ)



## srv\_putmsgq

**説明**                   メッセージ・キューにメッセージを入れます。

**構文**                   CS\_RETCODE srv\_putmsgq(msgqid, msgp, putflags)

SRV\_OBJID       msgqid;  
CS\_VOID         \*msgp;  
CS\_INT          putflags;

**パラメータ**           *msgqid*  
メッセージ・キューの識別子です。メッセージ・キューを名前で参照する場合は、`srv_getobjid` を呼び出して、名前を検索しメッセージ・キュー ID を返します。

*msgp*  
メッセージへのポインタです。メッセージ・データは、受信されて処理されるまでは有効でなければなりません。

*putflags*  
*putflags* の値は、論理和をとることができます。表 3-88 に、値とその意味を示します。

**表 3-88 : putflags の値 (srv\_putmsgq)**

値	説明
SRV_M_NOWAIT	このフラグが設定されると、メッセージがメッセージ・キューに置かれた直後に、 <code>srv_putmsgq</code> の呼び出しが戻る。
SRV_M_WAIT	SRV_M_WAIT が設定されると、メッセージが読み込まれるかキューが削除されるまでは <code>srv_putmsgq</code> は戻らない。
SRV_M_URGENT	このフラグが設定されると、メッセージは、メッセージ・キューのメッセージ・リストの最後ではなく先頭に置かれる。1つのキューに複数の緊急メッセージが追加された場合には、緊急メッセージは、キューに入った順でキューの先頭に表示される。

**戻り値**                   **表 3-89 : 戻り値 (srv\_putmsgq)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include      <ospublic.h>

/*
```

```

** Local Prototype.
*/
CS_RETCODE ex_srv_putmsgq PROTOTYPE((
SRV_OBJID  mqid,
CS_INT     flags
));
/*
** EX_SRV_PUTMSGQ
**
** Example routine to put a message into a message queue.
**
** Arguments:
** msgqid    Message queue identifier.
** putflags  Special instructions for srv_putmsgq.
**
** Returns:
**
** CS_SUCCEED
** CS_FAIL
*/
CS_RETCODE    ex_srv_putmsgq(mqid, flags)
SRV_OBJID    mqid;
CS_INT       flags;
{
    CS_CHAR    *msgp;
    msgqp = srv_alloc(20);
    strcpy(msgp, "Hi there");
    return(srv_putmsgq(mqid, msgp, flags));
}

```

#### 使用方法

- `srv_putmsgq` は、*msgp* のメッセージをメッセージ・キュー *msgqid* に入れます。
- メッセージは、ポインタとして渡されます。仮にメッセージを送信しているスレッドのコンテキストが変更されても、メッセージが指すデータは有効のままではなりません。

特に、メッセージを送信するスレッドのコンテキストにあるスタック・アドレスを指すメッセージを転送するときには、注意してください。これを行う場合には、メッセージがキューから削除されるまでは、メッセージを送ったスレッドがメッセージを送ったフレームから戻らないことを保証しなければなりません。この保証がないと、メッセージは他の目的に使用されているスタックを指すこともあります。

- `SRV_S_NUMMSGQUEUES` サーバ・プロパティは、Open Server アプリケーションが使用できるメッセージ・キューの数を決定します。「サーバ・プロパティ」(159 ページ)を参照してください。
- `SRV_S_MSGPOOL` サーバ・プロパティは、ランタイムに Open Server アプリケーションが使用できるメッセージの数を決定します。「サーバ・プロパティ」(159 ページ)を参照してください。

参照

[srv\\_createmsgq](#), [srv\\_deletemsgq](#), [srv\\_getmsgq](#), [srv\\_getobjid](#)

## srv\_realloc

説明

メモリの再割り付けを行います。

構文

```
CS_VOID* srv_realloc(mp,newsiz)
```

```
CS_VOID      *mp;
CS_INT       newsiz;
```

パラメータ

*mp*

古いメモリ・ブロックへのポインタです。

*newsiz*

再割り付けするバイト数です。

戻り値

表 3-90 : 戻り値 (srv\_realloc)

戻り値	意味
新しく割り付けられた領域へのポインタ	新しい領域のロケーション。
null ポインタ	Server-Library は <i>newsiz</i> バイトの割り付けができなかった。

例

```
#include      <ospublic.h>

/*
** Local Prototype.
*/
extern CS_RETCODE ex_srv_realloc PROTOTYPE((
CS_VOID      *mp,
CS_INT       newsiz
));

/*
```

```

** EX_SRV_REALLOC
**
**   Reallocate a memory chunk.
**
** Arguments:
**   mp           A pointer to existing memory block.
**   newsize     The new size of the memory block.
**
** Returns:
**   CS_SUCCEEDED      Memory was allocated successfully.
**   CS_FAIL           An error was detected.
*/
CS_RETCODE      ex_srv_realloc(mp, newsize)
CS_VOID         *mp;
CS_INT          newsize;
{
    mp = srv_realloc(mp, newsize);

    if(mp == (CS_VOID *)NULL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

#### 使用法

- `srv_realloc` は、動的にメモリの再割り付けを行います。
- `mp` で参照されているブロックのサイズを `newsize` に変更し、(移動された可能性もある) ブロックへのポインタを返します。
- `srv_realloc` を使用して割り付けたメモリは、すべて `srv_free` を呼び出すことで解放しなければなりません。
- 通常の C 言語のメモリ割り付けルーチンが使用されるようなところで、`srv_realloc` を使用します。
- 現行バージョンでは、`srv_realloc` は、C 言語のルーチンの `realloc` を呼び出します。しかし、Open Server アプリケーションでは、`srv_props` ルーチンを使用して独自のメモリ管理ルーチンをインストールすることができます。ユーザ・インストール・ルーチンのパラメータ転送規則は、`realloc` のものと同一でなければなりません。アプリケーションがユーザ・インストール・ルーチンを使用できるように設定されていない場合には、Open Server は `realloc` を呼び出します。

#### 参照

[srv\\_alloc](#), [srv\\_free](#), [srv\\_props](#)

## srv\_recvpasssthru

**説明** クライアントからプロトコル・パケットを受信します。

**構文** CS\_RETCODE srv\_recvpasssthru(spp, recv\_bufp, infop)

```
SRV_PROC      *spp;
CS_BYTE       **recv_bufp;
CS_INT        *infop;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*recv\_bufp*

受信したプロトコル・パケットを含んでいるバッファの開始アドレスを受ける CS\_BYTE ポインタへのポインタです。

*infop*

srv\_recvpasssthru が CS\_FAIL を返した場合に SRV\_I\_UNKNOWN に設定される CS\_INT へのポインタです。表 3-91 は、srv\_recvpasssthru が CS\_SUCCEED を返す場合に \*infop に返される可能性のある値を示します。

**表 3-91 : CS\_SUCCEED の値 (srv\_recvpasssthru)**

値	説明
SRV_I_PASSTHRU_MORE	プロトコル・パケットを読むことに成功。これはメッセージ・パケットの終わりではない。
SRV_I_PASSTHRU_EOM	このパケットは、メッセージ・パケットの終わりである。

**戻り値**

**表 3-92 : 戻り値 (srv\_recvpasssthru)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include          <ospublic.h>
/*
** Local prototype.
*/
CS_RETCODE       ex_srv_recvpasssthru PROTOTYPE((
CS_VOID          *spp
));
/*
** EX_SRV_RECVPASSTHRU
**
```

```

** Example routine to receive protocol packets from a client.
**
** Arguments:
**     spp      A pointer to an internal thread control structure.
**
** Returns:
**     CS_SUCCEED If we were able to receive the packets.
**     CS_FAIL If were unsuccessful at receiving the packets.
**
*/
CS_RETCODE  ex_srv_recvpasssthru(spp)
SRV_PROC    *spp;
{
    CS_RETCODE    result;
    CS_BYTE      *recvbuf;
    CS_INT       info;

    /*
    ** Read packets until we get the EOM flag.
    */
    do
    {
        result = srv_recvpasssthru(spp, &recvbuf, &info);
    }
    while (result == CS_SUCCEED && info == SRV_I_PASSTHRU_MORE);

    return (result);
}

```

#### 使用方法

- `srv_recvpasssthru` は、プロトコル・パケットを内容を解釈せずに受け取ります。
- 一度 `srv_recvpasssthru` が呼び出されると、これを呼び出したイベント・ハンドラは「パススルー・モード」に入ったこととなります。パススルー・モードは、*\*info* に `SRV_I_PASSTHRU_EOM` が返されるときに終了します。
- イベント・ハンドラがパススルー・モード状態の場合は、他の **Server-Library** ルーチンを読み出すことはできません。
- パススルー・モードでは、クライアントに対する `SRV_CONNECT` ハンドラは、`srv_getloginfo`、`ct_setloginfo`、`ct_getloginfo`、および `srv_setloginfo` を呼び出すことによって、クライアントとリモート・サーバがプロトコル・パケットのフォーマットをネゴシエートできるようにする必要があります。これによって、異なるプラットフォーム上で動作しているクライアントおよびリモート・サーバは、必要なデータ変換を行うことができます。

- `srv_recvpass thru` は、`SRV_START`、`SRV_CONNECT`、`SRV_STOP`、`SRV_DISCONNECT`、`SRV_URGDISCONNECT`、`SRV_ATTENTION` 以外のすべてのイベント・ハンドラで呼び出すことができます。
- 一度 `srv_recvpass thru` を呼び出すと、実行中のスレッドは `srv_senddone` を発行するまで、I/O を実行するルーチンを呼び出すことができなくなります。ネットワーク I/O は、`srv_recvpass thru` を実行するイベント・ハンドラで発生できません。

参照

[srv\\_getloginfo](#), [srv\\_sendpass thru](#), [srv\\_setloginfo](#)

## srv\_regcreate

説明

レジスタード・プロシージャの登録を完了します。

構文

`CS_RETCODE srv_regcreate(spp, infop)`

```
SRV_PROC  *spp;
CS_INT    *infop;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*infop*

`CS_INT` を指すポインタです。表 3-93 に、`srv_regcreate` が `CS_FAIL` を返す場合に *infop* に返される可能性がある値を示します。

表 3-93 : infop の値 (srv\_regcreate)

値	説明
<code>SRV_I_PEXISTS</code>	プロシージャはすでに登録されている。
<code>SRV_I_UNKNOWN</code>	その他のエラーが発生した。

戻り値

表 3-94 : 戻り値 (srv\_regcreate)

戻り値	意味
<code>CS_SUCCEED</code>	ルーチンが正常に終了した。
<code>CS_FAIL</code>	ルーチンが失敗。

例

```
#include <ospublic.h>

/*
** Local Prototype.
```

```

*/
CS_INT    ex_srv_regcreate PROTOTYPE((
SRV_PROC    *sproc
));

/*
** EX_SRV_REGCREATE
**   An example routine that completes the registration of a
**   registered procedure using srv_regcreate.
**
** Arguments:
**   sproc    A pointer to an internal thread control structure.
**
** Returns:
**   CS_SUCCEED    If the procedure was registered successfully.
**   CS_FAIL       If the supplied internal control structure is
**   **           NULL.
**   SRV_I_EXIST   If the procedure is already registered.
**   SRV_I_UNKNOWN If some other error occurred.
*/
CS_INT    ex_srv_regcreate(sproc)
SRV_PROC    *sproc;
{
    CS_INT    info;    /* The reason for failure */

    /*
    ** Check whether the internal control structure is NULL.
    */
    if ( sproc == (SRV_PROC *)NULL )
    {
        return((CS_INT)CS_FAIL);
    }

    /*
    ** Now register the procedure already defined by
    ** srv_regdefine and(or) srv_regparam.If an error
    ** occurred, return the cause of error.
    */
    if ( srv_regcreate(sproc, &info) == CS_FAIL )
    {
        return(info);
    }

    /* The procedure is registered.*/
    return((CS_INT)CS_SUCCEED);
}

```



- 使用法
- プロシージャを登録するために必要な情報がすべて提供された後に、`srv_regcreate` は登録を完了します。
  - プロシージャの名前とパラメータは、あらかじめそれぞれ `srv_regdefine` と `srv_regparam` を使用して定義されていなければなりません。
  - 一度登録されると、クライアント・アプリケーションからでも Open Server アプリケーション・プログラムの中からでも、プロシージャを呼び出すことができます。
  - プロシージャを登録する例は、[srv\\_regdefine](#) を参照してください。
- 参照 [srv\\_regdefine](#), [srv\\_regdrop](#), [srv\\_reglis](#), [srv\\_regparam](#)

## srv\_regdefine

説明 プロシージャを登録するプロセスを開始する。

構文

```
CS_RETCODE srv_regdefine(spp, procnamep,
                        namelen, funcp)

SRV_PROC      *spp;
CS_CHAR      *procnamep;
CS_INT       namelen;
SRV_EVENTHANDLE_FUNC(*funcp)();
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*procnamep*

プロシージャ名へのポインタです。

*namelen*

プロシージャ名の長さです。*\*proc\_namep* の文字列が `null` で終了する場合、*namelen* は `CS_NULLTERM` とすることもできます。

*funcp*

プロシージャが実行されるたびに呼び出される関数へのポインタです。このパラメータを `null` に設定すると、「ノーティフィケーション(通知)プロシージャ」が登録されます。ノーティフィケーション・プロシージャは、クライアント間での通信には便利です。「[レジスタード・プロシージャ](#)」(185 ページ)を参照してください。

戻り値

**表 3-95 : 戻り値 (srv\_regdefine)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>
#include <stdio.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_regdefine PROTOTYPE((
SRV_SERVER      *server
));
CS_RETCODE      stop_serv PROTOTYPE((
SRV_PROC        *spp
));

/*
** Local defines.
*/
#define  STOP_SERV    "stop_serv"

/*
** STOP_SERV
**   This function is called when the client sends the stop_serv
**   registered procedure.
**
** Arguments:
**   spp           A pointer to internal thread control structure.
**
** Returns:
**   SRV_CONTINUE
*/
CS_INT          stop_serv(spp)
SRV_PROC        *spp;
{
    /* Queue a SRV_STOP event.*/
    (CS_VOID)srv_log((SRV_SERVER *)NULL, CS_TRUE,
        "Stopping Server %Yn" , CS_NULLTERM);

    /* Send a final DONE to client to acknowledge the command.*/
    if (srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED,
        (CS_INT)0)
```

```

        == CS_FAIL)
    {
        fprintf(stderr, "srv_senddone failed¥n" );
    }

    /* Queue a SRV_STOP event to shut down the server.*/
    if (srv_event(spp, SRV_STOP, (CS_VOID *)NULL)
        == CS_FAIL)
    {
        fprintf(stderr, "Error queuing SRV_STOP event¥n" );
    }
    return(SRV_CONTINUE);
}

/*
** EX_SRV_REGDEFINE
**
** Example routine to illustrate the use of srv_regdefine to
** register a procedure.
**
** Arguments:
** server      A pointer to the Open Server control structure.
**
** Returns:
**
** CS_SUCCEED If procedure was registered successfully.
** CS_FAIL    If an error occurred in registering the
** procedure.
*/
CS_RETCODE      ex_srv_regdefine (server)
SRV_SERVER      *server;
{
    SRV_PROC      *spp;
    CS_INT        info;

    /* Create a thread.*/
    spp = srv_createproc(server);

    if (spp == (SRV_PROC *)NULL)
        return(CS_FAIL);

    /* Define the procedure.*/
    if (srv_regdefine(spp, STOP_SERV, CS_NULLTERM, stop_serv)
        == CS_FAIL)
        return(CS_FAIL);
}

```

```

/* Complete the registration.*/
if (srv_regcreate(spp, &info) == CS_FAIL)
    return(CS_FAIL);

/*
** Terminate the thread created here.We do not care about
** the return code from srv_termproc here.
*/
(CS_VOID)srv_termproc(spp);

return (CS_SUCCEED);
}

```

使用法

- `srv_regdefine` は、プロシージャを登録するプロセスの最初の手順です。プロシージャは一度登録されると、クライアントから、または Open Server アプリケーション・プログラムの中からも呼び出せます。
- `srv_regdefine` を呼び出した後、`srv_regparam` を使用してプロシージャのパラメータを定義します。
- プロシージャの登録プロセスは、`srv_regcreate` を呼び出して終了します。
- `procnamep` と同一の名前を持つレジスタード・プロシージャがある場合はエラーが検出され、`srv_regcreate` が呼び出されたときにレポートされます。
- すべての要求されたプロシージャは、`SRV_CONTINUE` を返します。

参照

[srv\\_regcreate](#), [srv\\_regdrop](#), [srv\\_reglist](#), [srv\\_regparam](#)

## srv\_regdrop

説明

プロシージャの登録を解除します。

構文

```

CS_RETCODE srv_regdrop(spp, procnamep,
                       namelen, info)

SRV_PROC   *spp;
CS_CHAR    *procnamep;
CS_INT     namelen;
CS_INT     *infp;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*procnamep*

プロシージャ名へのポインタです。

*namelen*

レジスタード・プロシージャ名の長さです。名前が `null` で終了する場合、*namelen* は `CS_NULLTERM` とすることもできます。

*infop*

`CS_INT` へのポインタです。`srv_regdrop` が `CS_FAIL` を返すと、次の値のいずれかにフラグが設定されます。

- `SRV_I_PNOTKNOWN` - プロシージャは登録されていない。
- `SRV_I_UNKNOWN` - その他のエラーが発生した。

戻り値

**表 3-96 : 戻り値 (srv\_regdrop)**

戻り値	意味
<code>CS_SUCCEED</code>	ルーチンが正常に終了した。
<code>CS_FAIL</code>	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local Prototype.
**/
CS_RETCODE      ex_srv_regdrop PROTOTYPE((
SRV_PROC                *spp,
CS_CHAR                *name,
CS_INT                 namelen,
CS_INT                 *infop
));

/*
** EX_SRV_REGDROP
**
** Example routine to unregister a registered procedure using
** srv_regdrop.
**
** Arguments:
** spp      A pointer to an internal thread control structure.
** name     The name of the registered procedure to drop.
** namelen  The length of the registered procedure name.
** infop    A return pointer to an integer containing more
**           descriptive error information if this routine
**           returns CS_FAIL.
**
```

```

** Returns:
** CS_SUCCEED   Registered procedure was successfully deleted.
** CS_FAIL     Registered procedure was not deleted or does
**             not exist.
*/
CS_RETCODE     ex_srv_regdrop(spp, name, namelen, infop)
SRV_PROC       *spp;
CS_CHAR        *name;
CS_INT         namelen;
CS_INT         *infop;
{
    /* Initialization.*/
    *infop = (CS_INT)0;
    /* Execute the procedure.*/
    if (srv_regdrop(spp, name, namelen, infop) != CS_SUCCEED)
    {
        /* Open Server has set infop to a specific error.*/
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

- 使用法
- `srv_regdrop` は、以前に `srv_regcreate` で登録したプロシージャを削除します。
  - このプロシージャの通知を待っているクライアント・スレッドには、プロシージャが削除されたことが通知されます。

参照 [srv\\_regcreate](#), [srv\\_regdefine](#), [srv\\_reglst](#), [srv\\_regparam](#)

## srv\_regexec

説明 レジスタード・プロシージャを実行します。

構文 `CS_RETCODE srv_regexec(spp, infop)`  
`SRV_PROC *spp;`  
`CS_INT *infop;`

パラメータ *spp*  
 内部スレッド制御構造体へのポインタです。

*infop*  
`CS_INT` を指すポインタです。表 3-97 に、`srv_regexec` が `CS_FAIL` を返す場合に *infop* に返される可能性がある値を示します。

表 3-97 : infop の値 (srv\_regexec)

値	説明
SRV_I_PNOTKNOWN	プロシージャは登録されていない。
SRV_I_PPARAMERR	パラメータ・エラーがある。
SRV_I_PNOTIFYERR	通知を送信中にエラーが発生した。

戻り値

表 3-98 : 戻り値 (srv\_regexec)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype
*/
CS_RETCODE      ex_srv_regexec PROTOTYPE((
SRV_PROC        *spp,
CS_INT          &infop
));

/*
** EX_SRV_REGEXEC
**
** Example routine to complete the execution of a registered
** procedure using srv_regexec. This routine should be called
** after srv_reginit and srv_regparam.
**
** Arguments:
** spp          A pointer to an internal thread control structure.
** infop        A return pointer to an integer containing more
**              descriptive error information if this routine
**              returns CS_FAIL.
**
** Returns:
** CS_SUCCEED   Registered procedure executed successfully.
** CS_FAIL      Registered procedure not executed, or
**              notifications not completed successfully.
*/
CS_RETCODE ex_srv_regexec(spp, infop)
SRV_PROC   *spp;
CS_INT     &infop;
{
```

```

/* Initialization.*/
&infop = (CS_INT)0;

/* Execute the procedure.*/
if (srv_regexec(spp, infop) != CS_SUCCEED)
{
    /*
    ** Open Server has set the argument to a specific
    ** error.
    */
    return(CS_FAIL);
}
return (CS_SUCCEED);
}

```

使用法

- `srv_regexec` は、レジスタード・プロシージャを実行します。
- `srv_regexec` を呼び出す前に、`srv_reginit` および `srv_regparam` でプロシージャ名とパラメータを指定します。

---

**警告！** Open Server のシステム・レジスタード・プロシージャは、最終の DONE を送信します。アプリケーションがイベント・ハンドラから `srv_regexec` を使用してシステム・レジスタード・プロシージャを実行した場合には、アプリケーションはイベント・ハンドラ・コードから最終の DONE を送信しません。この場合、Open Server はステータス・エラーとなります。

---

参照

[srv\\_reginit](#), [srv\\_regparam](#)

## srv\_reginit

説明

レジスタード・プロシージャの実行を開始します。

構文

```

CS_RETCODE srv_reginit(spp, procnamep,
                       namelen, options)

SRV_PROC    *spp;
CS_CHAR     *procnamep;
CS_INT      namelen;
unsigned short options;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。



*procnamep*

レジスタード・プロシージャ名へのポインタです。

*namelen*

プロシージャ名の長さです。名前が `null` で終了する場合、*namelen* は `CS_NULLTERM` とすることもできます。

*options*

どのスレッドに通知するかを決めるフラグです。表 3-99 に、*options* の有効値を示します。

表 3-99 : options の値 (srv\_reginit)

値	説明
SRV_M_PNOTIFYALL	通知リストにある待機中のすべてのスレッドに通知する。
SRV_M_PNOTIFYNEXT	待機時間が最長のスレッドにのみ通知する。

## 戻り値

表 3-100 : 戻り値 (srv\_reginit)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include          <ospublic.h>
/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_reginit  PROTOTYPE((
SRV_PROC        *sp,
CS_CHAR         *pname,
CS_INT          nlen
));

/*
** EX_SRV_REGINIT
**
** This routine demonstrates how to use srv_reginit to
** initiate the execution of a registered procedure.
**
** Arguments:
**     sp          A pointer to an internal thread control
                  structure.
**     pname      The name of the procedure to execute.
**     nlen       The length of the procedure name.
** Returns
```

```

**
**      CS_SUCCEED      If the registered procedure began execution.
**      CS_FAIL        If an error was detected.
**
*/
CS_RETCODE      ex_srv_reginit(sp, pname, nlen)
SRV_PROC        *sp;
CS_CHAR         *pname;
CS_INT          nlen;
{
    /*
    ** Call srv_reginit to initiate the execution of this
    ** registered procedure; ask that all threads waiting for
    ** notification of this event be notified.
    */
    if( srv_reginit(sp, pname, nlen, SRV_M_PNOTIFYALL) ==
        CS_FAIL )
    {
        /*
        ** An error was already raised.
        */
        return CS_FAIL;
    }

    /*
    ** All done.
    */
    return CS_SUCCEED;
}

```

#### 使用法

- `srv_reginit` はレジスタード・プロシージャの実行プロセスの最初の手順です。
- `srv_reginit` を呼び出した後に、`srv_regparam` を使ってプロシージャのパラメータを定義します。
- レジスタード・プロシージャの実行は、`srv_regexec` を呼び出して行います。
- プロシージャが存在しない場合には、`srv_regexec` は、エラーの検出とレポートを行います。
- レジスタード・プロシージャが実行されると、**Open Server** は、プロシージャの通知リスト内のスレッドに通知します。`options` パラメータを指定して、リスト中のすべてのスレッドに通知を送信するか、待機時間が最長のスレッドにだけ通知を送信するかを決定します。

- Open Server アプリケーションでは、レジスタード・プロシージャを最大 16 レベルまでネストすることができます。

参照 [srv\\_regexec](#), [srv\\_regparam](#)

## srv\_reglist

**説明** Open Server に登録されているすべてのプロシージャのリストを取得します。

**構文** CS\_RETCODE srv\_reglist(spp, proclistp)

```
SRV_PROC      *spp;
SRV_PROCLIST **proclistp;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*proclistp*

結果を含む SRV\_PROCLIST のアドレスに設定される SRV\_PROCLIST ポインタへのポインタです。srv\_reglist が呼び出されるとときに、Open Server はこの構造体用の領域を割り付けます。

**戻り値**

**表 3-101 : 戻り値 (srv\_reglist)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include      <ospublic.h>
/*
** Local Prototype.
*/
CS_RETCODE    ex_srv_reglist PROTOTYPE((
SRV_PROC      *spp,
SRV_PROCLIST **proclp
));

/*
** EX_SRV_REGLIST
**
** Arguments:
**
**   spp      Pointer to an internal thread control structure.
```

```

**  proclp      Pointer to a SRV_PROCLIST pointer that will be set
**              to point to the result.
**
** Returns
**
**  CS_SUCCEED   srv_reglist was successful.
**  CS_FAIL     An argument was invalid or srv_reglist failed.
**
*/
CS_RETCODE      ex_srv_reglist (spp, proclp)
SRV_PROC        *spp;
SRV_PROCLIST    **proclp;
{
    /* Check arguments.*/
    if (spp == (SRV_PROC *)NULL)
    {
        return(CS_FAIL);
    }
    return(srv_reglist(spp,proclp));
}

```

**使用法**

- `srv_reglist` は、スレッドに対する、現在のレジスタード・プロシージャのリストをすべて返します。
- `proclstp` パラメータは、Open Server によって割り付けられ、初期化される構造体を指すように設定されます。SRV\_PROCLIST 構造体は次のように定義されます。

```

typedef struct srv__proclist
{
    CS_INT      num_procs;      /* The number of procedures */
    CS_CHAR     **proc_list;    /* Array of procedure names */
} SRV_PROCLIST;

```

- SRV\_PROCLIST 構造体は、不要になった場合、`srv_reglistfree` を使って割り付けを解除します。

**参照**

[srv\\_reglistfree](#)

## srv\_reglistfree

**説明**

以前に割り付けられた SRV\_PROCLIST 構造体を解放します。

**構文**

CS\_RETCODE `srv_reglistfree(spp, proclstp)`

SRV\_PROC            \*spp;  
 SRV\_PROCLIST        \*proclistp;

パラメータ            *spp*  
                       内部スレッド制御構造体へのポインタです。

*proc\_list*  
                       以前に *srv\_reglist* または *srv\_regwatchlist* を使用して割り付けられた  
                       SRV\_PROCLIST 構造体へのポインタです。

戻り値

表 3-102 : 戻り値 (*srv\_reglistfree*)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype
*/
CS_RETCODE       ex_srv_reglistfree PROTOTYPE((
SRV_PROC         *srvproc,
SRV_PROCLIST     *reglistp
));

/*
** EX_SRV_REGLISTFREE
**
**   Example routine to free a previously allocated reglist.
**
** Arguments:
**   srvproc   A pointer to an internal thread control structure.
**   reglistp  A pointer to the list to free.
**
** Returns:
**
**   CS_SUCCEED
**   CS_FAIL
**
*/
CS_RETCODE       ex_srv_reglistfree(srvproc, reglistp)
SRV_PROC         *srvproc;
SRV_PROCLIST     *reglistp;
{
    return(srv_reglistfree(srvproc, reglistp));
}
```

**使用法** `srv_reglistfree` は、`srv_reglist` または `srv_regwatchlist` を使用して割り付けられた `SRV_PROCLIST` 構造体の割り付けを解除します。

**参照** [srv\\_reglist](#), [srv\\_regwatchlist](#)

## srv\_regnowatch

**説明** レジスタード・プロシージャの通知リストから、クライアント・スレッドを削除する。

**構文** `CS_RETCODE` `srv_regnowatch(spp, procnamep, namelen, infop)`

```
SRV_PROC   *spp;
CS_CHAR    *procnamep;
CS_INT     namelen;
CS_INT     *infop;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*procnamep*

プロシージャ名へのポインタです。

*namelen*

プロシージャ名の長さです。名前が `null` で終了する場合、*namelen* は `CS_NULLTERM` とすることもできます。

*infop*

`CS_INT` を指すポインタです。[表 3-103](#) に、`srv_regnowatch` が `CS_FAIL` を返す場合に *infop* に返される可能性がある値を示します。

表 3-103 : infop の値 (srv\_regnowatch)

値	説明
SRV_I_PNOTCLIENT	非クライアント・スレッドが指定された。
SRV_I_PNOTKNOWN	プロシージャは、Open Server アプリケーションには認識されていない。
SRV_I_PNOPENDING	そのスレッドは、このプロシージャの通知リストには含まれていない。
SRV_I_PPARAMERR	パラメータ・エラーが生じた。
SRV_I_UNKNOWN	その他のエラーが発生した。

戻り値

表 3-104 : 戻り値 (srv\_regnowatch)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype.
*/
extern CS_RETCODE  ex_srv_regnowatch PROTOTYPE((
CS_VOID          *spp,
CS_CHAR          *procnamep,
CS_INT           namelen
));

/*
** EX_SRV_REGNOWATCH
**
**   Remove a client thread from the notification list for the
**   specified registered procedure.
**
** Arguments:
**   spp           A pointer to an internal thread control
**                structure.
**   procnamep    A pointer to the name of the registered
**                procedure.
**   namelen      The length of the registered procedure name.
**
** Returns:
**   CS_SUCCEED   The thread was removed from notification list.
**   CS_FAIL      An error was detected.
*/
```

```

CS_RETCODE      ex_srv_regnowatch(spp, procnamep, namelen)
SRV_PROC        *spp;
CS_CHAR         *procnamep;
CS_INT          namelen;
{
    if(srv_regnowatch(spp, procnamep, namelen, (CS_INT *)NULL)
        == CS_FAIL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

- 使用法
- `srv_regnowatch` は、指定したプロシージャを実行したときに、通知するスレッドのリストからクライアント・スレッドを削除します。
  - プロシージャ名の最大長は `SRV_MAXNAME` です。

参照 [srv\\_regwatch](#), [srv\\_regwatchlist](#)

## srv\_regparam

説明 定義されているレジスタード・プロシージャに対してパラメータを記述する、またはレジスタード・プロシージャの実行に対してデータを提供する。

構文

```

CS_RETCODE srv_regparam(spp, param_namep, namelen,
                        type, datalen, datap)

SRV_PROC    *spp;
CS_CHAR     *param_namep;
CS_INT      namelen;
CS_INT      type;
CS_INT      datalen;
CS_BYTE     *datap;

```

パラメータ *spp* 内部スレッド制御構造体へのポインタです。

*param\_namep* パラメータ名へのポインタです。プロシージャを登録するときには、このパラメータは必須条件です。プロシージャを呼び出すときに、プロシージャが登録されたときのパラメータの定義の順序と同じ順序でパラメータを指定する場合は、このパラメータを `null` にすることもできます。



*namelen*

パラメータ名の長さです。 *param\_namep* が null で終了する場合、 *namelen* は CS\_NULLTERM とすることもできます。

*type*

パラメータのデータ型です。 Open Server データ型のリストについては、「[データ型](#)」(229 ページ)を参照してください。

*datalen*

パラメータのデータ長です。固定長データ型の場合、このパラメータは無視されます。 null データ値を示すには、 *datalen* を 0 に設定します。クライアントがパラメータ値を指定しない場合には、Open Server アプリケーションはデフォルト値の長さを設定します。デフォルト値を定義したくない場合には、 *datalen* を SRV\_NODEFAULT に設定してください。

*datap*

データへのポインタです。プロシージャを登録する場合は、 *\*datap* の値はプロシージャの今後の呼び出しのデフォルト値を示します。プロシージャを呼び出している場合、デフォルト値を受け取るには *datap* を NULL に設定します。

## 戻り値

表 3-105 : 戻り値 (srv\_regparam)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include      <ospublic.h>

/*
** Local prototype.
*/
CS_RETCODE  ex_srv_regparam PROTOTYPE((
SRV_PROC    *spp
));

/*
** Local defines.
*/
#define PARAMNAME (CS_CHAR *)" myparam" /* Parameter name.*/

#define PARAMDEFAULT (CS_INT)100

/*
```

```

**The default value for the parameter.
*/

#define PARAMVAL (CS_INT)20 /* The value for this invocation.*/

/*
** EX_SRV_REGPARAM
**
** Example routine to describe a parameter for a registered
** procedure.
**
** Arguments:
** spp      A pointer to an internal thread control structure.
**
** Returns:
** CS_SUCCEED If we were able to describe the parameter.
** CS_FAIL    If an error was detected.
*/
CS_RETCODE      ex_srv_regparam(spp)
SRV_PROC        *spp;
{
    CS_RETCODE    result;
    CS_INT        param;

    /* Define the parameter with a default.*/
    param = PARAMDEFAULT;
    result = srv_regparam(spp, PARAMNAME, CS_NULLTERM,
        CS_INT_TYPE, sizeof(CS_INT), (CS_BYTE *)&param);

    if (result == CS_FAIL)
    {
        return(CS_FAIL);
    }

    /* Define the parameter with no default.*/
    result = srv_regparam(spp, PARAMNAME, CS_NULLTERM,
        CS_INT_TYPE, SRV_NODEFAULT, (CS_BYTE *)NULL);

    if (result == CS_FAIL)
    {
        return(CS_FAIL);
    }

    /* Give a non-default value for the parameter.*/
    param = PARAMVAL;
    result = srv_regparam(spp, PARAMNAME, CS_NULLTERM,

```

```

        CS_INT_TYPE, sizeof(CS_INT), (CS_BYTE *)&param);

    return (result);
}

```

#### 使用法

- `srv_regparam` は、プロシージャの呼び出しまたは登録のためのプロシージャ・パラメータを指定します。`srv_regparam` は、`srv_reginit` または `srv_regdefine` を呼び出してから実行します。
- レジスタード・プロシージャは、最大 1024 のパラメータを持つことができます。
- プロシージャを登録するときには、`srv_regparam` を使って、プロシージャのパラメータとデフォルト値のプロパティを定義します。
- プロシージャを呼び出すときには、デフォルト値が指定されているパラメータを除き、各パラメータについて `srv_regparam` を呼び出します。
- `null` データ値を示すには、`datalen` を 0 に設定します。
- プロシージャを実行するときにパラメータのデフォルト値を受け取るには、`datap` を `NULL` に設定します。
- パラメータに提供されたデフォルト値をプロシージャの実行に使用する場合は、`srv_regparam` を呼び出す必要はありません。

#### 参照

[srv\\_regcreate](#), [srv\\_regdefine](#), [srv\\_reginit](#), [srv\\_regexec](#), 「データ型」(229 ページ)

## srv\_regwatch

#### 説明

指定されたプロシージャの通知リストに、クライアント・スレッドを追加する。

#### 構文

```

CS_RETCODE srv_regwatch(spp, proc_namep, namelen,
                        options, infop)

SRV_PROC      *spp;
CS_CHAR      *proc_namep;
CS_INT       namelen;
CS_INT       options;
CS_INT       *infop;

```

#### パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*proc\_namep*

プロシージャの名前です。

*namelen*

プロシージャ名の長さです。プロシージャ名が null で終了する場合、*namelen* は CS\_NULLTERM とすることもできます。

*options*

一度かぎりの通知要求なのか、永続的な通知要求なのかを指定するフラグです。表 3-106 に、*options* の有効値を示します。

**表 3-106 : options の値 (srv\_regwatch)**

値	説明
SRV_NOTIFY_ONCE	一度目の通知の後に、クライアント・スレッドはプロシージャの通知リストから削除される。
SRV_NOTIFY_ALWAYS	srv_regnowatch を使ってプロシージャの通知リストからスレッドが削除されるまで、プロシージャが実行されるたびにクライアント・スレッドは通知を受ける。

*infop*

表 3-107 に、srv\_regwatch が CS\_FAIL を返す場合に \**infop* に返される可能性がある値を示します。

**表 3-107 : infop の値 (srv\_regwatch)**

値	説明
SRV_I_PNOTKNOWN	プロシージャは、Open Server アプリケーションには認識されていない。スレッドは通知リストに追加されなかった。
SRV_I_PINVOPT	有効でない <i>options</i> 値が指定された。スレッドは通知リストに追加されなかった。
SRV_I_PNOTCLIENT	非クライアント・スレッドが指定された。スレッドは通知リストに追加されなかった。
SRV_I_PNOTIFYEXISTS	スレッドは、指定されたプロシージャの通知リストにすでに存在する。

戻り値

**表 3-108 : 戻り値 (srv\_regwatch)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
```

```

/*
** Local Prototype.
*/
CS_INT      ex_srv_regwatch PROTOTYPE((
SRV_PROC    *sproc,
CS_CHAR     *procedure_name
));

/*
** EX_SRV_REGWATCH
**   An example routine to add a client thread to the
**   notification list for a specified procedure.
**
** Arguments:
**
**   sproc           A pointer to an internal thread control
**   structure.
**   procedure_name  The null terminated procedure name.
**
** Returns:
**   CS_SUCCEEDED      If the thread was added to the
**                     notification list.
**   SRV_I_PNOTKNOWN   The procedure is not known to the Open
**   Server application.
**   SRV_I_PNOTCLIENT  A non-client thread was specified.
**   SRV_I_PNOTIFYEXISTS The thread is already on the
**                     notification list for the specified
**   procedure.
**   CS_FAIL           The attempt to add the thread to the
**                     notification failed due to other
**                     errors.
*/
CS_INT      ex_srv_regwatch(sproc, procedure_name)
SRV_PROC    *sproc;
CS_CHAR     *procedure_name;
{
    CS_INT      info;

    if ( srv_regwatch(sproc, procedure_name, CS_NULLTERM,
                     SRV_NOTIFY_ALWAYS, &info) == CS_FAIL )
    {
        if ( (info == SRV_I_PNOTKNOWN)
            || (info == SRV_I_PNOTCLIENT)
            || (info == SRV_I_PNOTIFYEXISTS) )
        {
            return(info);
        }
    }
}

```

```

    }
    else
    {
        return((CS_INT)CS_FAIL);
    }
}

return((CS_INT)CS_SUCCEEDED);
}

```

使用法

- `srv_regwatch` は、指定されたプロシージャが実行したときに通知するスレッドのリストに、スレッドを追加します。
- `options` フラグでは、プロシージャ実行時に必ずスレッドに通知するか、次回のプロシージャ実行時に 1 回だけ通知するかを指定します。
- 通知要求を取り消すには、`srv_regnowatch` を使用します。

参照

[srv\\_regnowatch](#), [srv\\_regwatchlist](#)

## srv\_regwatchlist

説明

クライアント・スレッドが通知要求待ちになっている、すべてのレジスタード・プロシージャのリストを返します。

構文

```

CS_RETCODE srv_regwatchlist(spp, proclistp)
SRV_PROC   *spp;
SRV_PROCLIST **proclistp;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*proclistp*  
レジスタード・プロシージャの数と各レジスタード・プロシージャの名前を持つ構造体へのポインタを指すポインタです。Open Server は、この構造体用に領域を割り付けます。

戻り値

表 3-109 : 戻り値 (srv\_regwatchlist)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_regwatchlist PROTOTYPE((
SRV_PROC        *spp
));

/*
** EX_SRV_REGWATCHLIST
**
** Example routine to get a list of all registered procedures
** for which a client thread has notifications pending.
**
** Arguments:
** spp          A pointer to an internal thread control structure.
**
** Returns:
**
** CS_SUCCEED   The list returned successfully.
** CS_FAIL      An error was detected.
*/
CS_RETCODE      ex_srv_regwatchlist(spp)
SRV_PROC        *spp;
{
    SRV_PROCLIST *listp;

    if (srv_regwatchlist(spp, &listp) == CS_FAIL)
        return(CS_FAIL);

    /*
    ** Process the information in the list and free the
    ** memory allocated for the list.
    */
    (CS_VOID)srv_reglistfree(spp, listp);

    return (CS_SUCCEED);
}
```

使用法

- `srv_regwatchlist` は、クライアント・スレッドが通知を要求したレジスタード・プロシージャのリストを返します。
- `proclistp` パラメータは、Open Server によって割り付けられ、初期化される `SRV_PROCLIST` 構造体を指します。SRV\_PROCLIST 構造体を次に示します。

```
typedef struct srv_proclist
{
    CS_INT      num_procs;      /* The number of procedure names */
    CS_CHAR     **proc_list;    /* The list of procedure names */
} SRV_PROCLIST;
```

- アプリケーションは、`srv_reglistfree` を呼び出すことによって、SRV\_PROCLIST 構造体の割り付けを解除します。

参照

[srv\\_reglistfree](#)

## srv\_rpcdb

説明

現在のリモート・プロシージャの対象のデータベース要素を返します。

構文

```
CS_CHAR      *srv_rpcdb(spp, lenp)
SRV_PROC     *spp;
CS_INT       *lenp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*lenp*

データベース名の長さが格納されている *int* 変数へのポインタです。  
*lenp* には NULL も指定できますが、その場合にはデータベース名の長さは返されません。



戻り値

表 3-110: 戻り値 (srv\_rpcdb)

戻り値	意味
現在の RPC の名前のデータベース要素を含む、null で終了する文字列を指すポインタ (CS_CHAR *) NULL	現在の RPC の名前のデータベース要素のロケーション。
	現在の RPC が存在しない。 Open Server は <i>lenp</i> を -1 に設定し、情報エラーが発生する。

例

```
#include          <ospublic.h>
/*
** Local Prototype.
**
CS_RETCODE       ex_srv_rpcdb PROTOTYPE((
SRV_PROC         *spp,
CS_CHAR          **dbp,
CS_INT           *lenp
));
/*
** EX_SRV_RPCDB
**
** Example routine to return the database component name of the
** current remote procedure call designation, using srv_rpcdb.
**
** Arguments:
**     spp      A pointer to an internal thread control structure.
**     dbp     A return pointer to the null terminated database name.
**     lenp    A return pointer to an integer containing the length
**             of the database name.
**
** Returns:
**     CS_SUCCEEDED Database component name returned successfully.
**     CS_FAIL       An error was detected.
**
CS_RETCODE       ex_srv_rpcdb(spp, dbp, lenp)
SRV_PROC         *spp;
CS_CHAR          **dbp;
CS_INT           *lenp;
{
    /* Initialization.*/
    *lenp = (CS_INT)0;
    /* Retrieve the database component name.*/
    if ((*dbp = (CS_CHAR *)srv_rpcdb(spp, lenp)) == (CS_CHAR
        *)NULL)
```

```

    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

使用法

- `srv_rpcdb` は `CS_CHAR` ポインタを返します。これは、最後が `null` で終了する文字列へのポインタです。文字列には、現在のリモート・プロシージャ・コールの名前のデータベース名に当たる部分が含まれています。
- `srv_rpcdb` は、RPC の名前からデータベース名に当たる部分だけを返します。所有者や RPC 番号を示すオプションの指定子など、データベース名以外の部分は返しません。完全に修飾されたストアド・プロシージャの名前の形式は、*database.owner.rpcname;number* です。RPC の名前から他の部分 (存在する場合) を取得するには、`srv_rpcname`、`srv_rpcowner`、`srv_rpcnumber` を使用してください。

参照

[srv\\_numparams](#), [srv\\_rpcname](#), [srv\\_rpcnumber](#), [srv\\_rpcoptions](#), [srv\\_rpcowner](#)

## srv\_rpcname

説明

現在のリモート・プロシージャ・コールの名前から、RPC 名に当たる部分を返します。

構文

```

CS_CHAR *srv_rpcname(spp, lenp)
SRV_PROC *spp;
CS_INT *lenp;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*lenp*  
RPC 名の長さが格納されているバッファへのポインタです。*lenp* には NULL も指定できますが、その場合には RPC 名の長さは返されません。

戻り値

表 3-111 : 戻り値 (srv\_rpcname)

戻り値	意味
現在の RPC の名前のデータベース要素を含む、null で終了する名前要素を指すポインタ	現在の RPC の名前のデータベース要素のロケーション。
null ポインタ	現在の RPC が存在しない。 Open Server は <i>lenp</i> を -1 に設定し、情報エラーが発生する。

例

```
#include          <ospublic.h>
/*
** Local Prototype.
*/
CS_RETCODE       ex_srv_rpcname PROTOTYPE((
SRV_PROC        *sp,
CS_CHAR         *buf,
CS_INT          buflen,
CS_INT          *lenp
));

/*
** EX_SRV_RPCNAME
**
** This routine demonstrates how to use srv_rrpcname to obtain
** the name of the remote procedure call received by this
** thread.
**
** Arguments:
**     sp          A pointer to an internal thread control
**     structure.
**     buf         The address of the buffer in which the RPC
**                 name will be returned.
**     buflen     The size of the name buffer.
**     lenp       The address of an integer variable, which
**                 will be set to the length of the name
**     returned.
**
** Returns
**     CS_SUCCEEDED   If the RPC name is returned.
**     CS_FAIL       If an error occurred.
*/
CS_RETCODE       ex_srv_rpcname(sp, buf, buflen, lenp)
SRV_PROC        *sp;
CS_CHAR         *buf;
```

```

CS_INT          buflen;
CS_INT          *lenp;
{
CS_CHAR          *np;    /* The procedure name pointer.*/

    /*
    ** Initialization.
    */
    np = (CS_CHAR *)NULL;
    *lenp = (CS_INT)0;

    /*
    ** Get the procedure name.
    */
    np = srv_rpcname(sp, lenp);

    if( np == (CS_CHAR *)NULL )
    {
        /*
        ** An error was already raised.
        */
        return CS_FAIL;
    }

    /*
    ** Copy the RPC name to the output buffer.
    */
    (void)strncpy(buf, np, buflen);

    /*
    ** All done.
    */
    return CS_SUCCEED;
}

```

**使用法**

- `srv_rpcname` は `CS_CHAR` ポインタを返します。これは、現在のリモート・プロシージャ・コール (“RPC”) の名前の RPC 名に当たる部分を含む、`null` で終了する文字列へのポインタです。
- `srv_rpcname` は RPC 名だけを返します。データベース、所有者、RPC 番号を示すオプションの指定子など、RPC 名以外の部分は返しません。たとえば、Adaptive Server Enterprise では、RPC 用の完全に修飾されたオブジェクト名は、`database.owner.rpcname;number` です。RPC の名前から他の部分 (存在する場合) を取得するには、`srv_rpcdb`、`srv_rpcowner`、`srv_rpcnumber` を使用してください。

- ユーザは、`srv_rpcname` を呼び出すことによって、RPC が存在するかどうかを調べることができます。RPC が存在しない場合には、Open Server は `SRV_ENORPC` エラーを返します。ユーザは、このエラーが検出された場合に、自分のエラー・ハンドラがこのエラーを無視するようにコーディングすることができます。

参照

[srv\\_numparams](#), [srv\\_rpcdb](#), [srv\\_rpcnumber](#), [srv\\_rpcoptions](#), [srv\\_rpcowner](#)

## srv\_rpcnumber

説明 現在のリモート・プロシージャで指定されている番号要素を返します。

構文 `CS_INT srv_rpcnumber(spp)`  
`SRV_PROC spp;`

パラメータ

*spp*  
 内部スレッド制御構造体へのポインタです。

戻り値

**表 3-112: 戻り値 (srv\_rpcnumber)**

戻り値	意味
0でない整数	現在のRPCの名前の番号要素。
-1	現在のRPCが存在しない。 Open Server は <i>lenp</i> を -1 に設定し、情報エラーが発生する。
0	クライアントがRPCを呼び出したときに番号要素を使用しなかった。

例

```
#include <ospublic.h>
/*
** Local Prototype.
*/
CS_INT ex_srv_rpcnumber PROTOTYPE((
SRV_PROC spp
));

/*
** EX_SRV_RCPNUMBER
**
** Example routine to show hiw to get the number of the
** current RPC designation.
**
```

```

**
** Arguments:
**
**     spp        A pointer to an internal thread control structure.
**
** Returns:
**
**     The number component of the current RPC' s designation.If
**     the client used no number component when it invoked the
**     RPC, 0 is returned.If there is not a current RPC, -1 is
**     returned and Open Server raises an informational error.
**
*/
CS_INT     ex_srv_rpcnumber(spp)
SRV_PROC   *spp;
{
    /* Check arguments.*/
    if (spp == (SRV_PROC *)NULL)
    {
        return (-1);
    }
    return((CS_INT)srv_rpcnumber(spp));
}

```

使用方法

- `srv_rpcnumber` は、現在のリモート・プロシージャ・コール (“RPC” ) の名前の番号要素を返します。
- `srv_rpcnumber` は、RPC の名前から番号に当たる部分だけを返します。所有者や RPC 名を示すオプションの指定子など、番号以外の部分は返しません。RPC の完全に修飾された名前は、*database.owner.rpcname;number* となります。RPC の名前から他の部分 ( 存在する場合 ) を取得するには、`srv_rpcname`、`srv_rpcowner`、`srv_rpcdb` を使用してください。

参照

[srv\\_numparams](#), [srv\\_rpcdb](#), [srv\\_rpcname](#), [srv\\_rpcoptions](#), [srv\\_rpcowner](#)

## srv\_rpcoptions

説明

現在のリモート・プロシージャ・コールのランタイム・オプションを返します。

構文

```

CS_INT     srv_rpcoptions(spp)
SRV_PROC   *spp;

```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

戻り値

表 3-113: 戻り値 (srv\_rpcoptions)

戻り値	意味
現在の RPC のためのランタイム・フラグを含む 0 でない整数	現在の RPC のランタイム・フラグ。
0	現在の RPC が存在しない。 Open Server はエラーになる。

例

```
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_rpcoptions PROTOTYPE((
SRV_PROC        *spp
));

/*
** EX_SRV_RPCOPTIONS
**
**      Example routine to retrieve RPC runtime options
**
** Arguments:
**      spp      A pointer to an internal thread control structure.
**
** Returns:
**
**      CS_SUCCEED
**      CS_FAIL
**
*/
CS_RETCODE      ex_srv_rpcoptions(spp)
SRV_PROC        *spp;
{
    CS_INT        options;

    if ( (options = srv_rpcoptions(spp)) == 0 )
        return(CS_FAIL);

    return (CS_SUCCEED);
}
```

- 使用法**
- `srv_rpcoptions` は、現在のレジスタード・プロシージャ・コールのランタイム・フラグを含む `CS_INT` 値を返します。
  - 現在は、`SRV_PARAMRETURN` が唯一のフラグです。  
`SRV_PARAMRETURN` が `CS_TRUE` の場合には、RPC は実行される前に再コンパイルされなければなりません。これは、RPC が `Adaptive Server Enterprise` で実行しているストアド・プロシージャである場合にのみ重要です。
- 参照** [srv\\_numparams](#), [srv\\_rpcdb](#), [srv\\_rpcname](#), [srv\\_rpcnumber](#), [srv\\_rpcowner](#)

## srv\_rpcowner

**説明** 現在のリモート・プロシージャ・コールで指定されている所有者要素を返します。

**構文**

```
CS_CHAR *srv_rpcowner(spp, lenp)
SRV_PROC  *spp;
CS_INT    *lenp;
```

**パラメータ**

*spp*  
内部スレッド制御構造体へのポインタです。

*lenp*  
所有者名の長さが格納されているバッファへのポインタです。*lenp* には `NULL` も指定できますが、その場合にはデータベース所有者の長さは返されません。

**戻り値** **表 3-114 : 戻り値 (srv\_rpcowner)**

戻り値	意味
現在の RPC の名前の <code>null</code> で終了する所有者要素を指すポインタ	現在の RPC の名前のデータベース要素のロケーション。
<code>null</code> ポインタ	現在の RPC が存在しない。 <code>Open Server</code> は <i>lenp</i> を <code>-1</code> に設定し、情報エラーが発生する。

**例**

```
#include <ospublic.h>

/*
** Local Prototype.
*/
```



```

extern CS_RETCODE    ex_srv_rpcowner PROTOTYPE((
CS_VOID            *spp,
CS_CHAR            *ownerp
));

/*
** EX_SRV_RPCOWNER
**
** Determine the owner component of an RPC destination.
**
** Arguments:
** spp            A pointer to an internal thread control structure.
** ownerp        A pointer to the buffer to which Open Server
**               returns the owner component.
**
** Returns:
** CS_SUCCEED    Owner component returned successfully.
** CS_FAIL       An error was detected.
*/
CS_RETCODE    ex_srv_rpcowner(spp, ownerp)
SRV_PROC      *spp;
CS_CHAR      *ownerp;
{
    CS_INT    len;

    ownerp = srv_rpcowner(spp, &len);

    if(len == (CS_INT)(-1))
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

**使用法**

- `srv_rpcowner` は `CS_CHAR` ポインタを返します。これは、現在のリモート・プロシージャ・コール (“RPC”) の名前の所有者に当たる部分を含む、`null` で終了する文字列へのポインタです。
- `srv_rpcowner` は、RPC の名前から所有者に当たる部分だけを返します。データベース名や RPC 番号を示すオプションの指定子など、所有者以外の部分は返しません。RPC の完全に修飾された名前は、`database.owner.rpcname;number` となります。RPC の名前から所有者以外の部分 (存在する場合) を取得するには、`srv_rpcname`、`srv_rpcdb`、`srv_rpcnumber` を使用してください。

**参照**

[srv\\_numparams](#)、[srv\\_rpcdb](#)、[srv\\_rpcname](#)、[srv\\_rpcnumber](#)、[srv\\_rpcoptions](#)

## srv\_run

**説明** Open Server アプリケーションを開始します。

**構文** CS\_RETCODE srv\_run(ssp)  
 SRV\_SERVER \*ssp;

**パラメータ** ssp  
 Open Server の制御構造体へのポインタです。この引数はオプションです。

**戻り値** **表 3-115 : 戻り値 (srv\_run)**

戻り値	意味
CS_SUCCEED	サーバが停止される。
CS_FAIL	Open Server がサーバを起動できなかった。 srv_run が CS_FAIL を返した場合、アプリケーションは、 srv_init を呼び出してから、srv_run を再度呼び出します。

**例**

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_run PROTOTYPE((CS_VOID));

/*
** EX_SRV_RUN
** An example routine to start up an Open Server using srv_run.
**
** Arguments:
** None.
**
** Returns:
** SRV_STOP If the server is stopped.
** CS_FAIL If the server can't be brought up.
*/
CS_RETCODE ex_srv_run()
{
    return(srv_run((SRV_SERVER *)NULL));
}
```

**使用法**

- srv\_run は、Open Server アプリケーションを起動、あるいは再起動します。

- `srv_run` は、サーバが `SRV_STOP` イベントで停止したときに戻ります。
- 一度起動すると、サーバはクライアント要求を受信し、その要求を処理するために定義された関数を呼び出し、さらに要求を受信します。
- サーバが停止した場合には、`srv_init` を使って再初期化してから、再起動しなければなりません。

---

**注意** `srv_run` が DLL のエントリ関数で呼び出された場合、デッドロックが起きることがあります。`srv_run` はオペレーティング・システム・スレッドを作成し、システム・ユーティリティを使用して同期化しようとしています。この同期化は、オペレーティング・システムの直列化プロセスと競合します。

---

参照

[srv\\_init, srv\\_props, 「イベント」 \(100 ページ\)](#)

## srv\_s\_ssl\_local\_id

説明	ローカル ID (認証) ファイルへのパスを指定するために使用されるプロパティです。
構文	<pre>typedef struct _cs_sslid {     CS_CHAR    *identity_file;     CS_CHAR    *identity_password; } CS_SSLIDENTITY</pre>
パラメータ	<p><i>identity_file</i> デジタル証明書およびそれに関連付けられたプライベート・キーを含むファイルへのパスを提供します。</p> <p><code>CS_GET</code> は、<code>CS_CONNECTION</code> に設定されている場合にのみ、使用されている <i>identity_file</i> を返します。</p> <p><i>identity_password</i> プライベート・キーを暗号化するのに使用されます。</p>

## srv\_select (UNIX のみ)

**説明** 指定の I/O オペレーションに対してファイル記述子の準備ができてい  
るかどうかをチェックします。

**構文** CS\_INT srv\_select(nfds, &readmaskp, writemaskp,  
exceptmaskp, waitflag)

```
CS_INT          nfds;
SRV_MASK_ARRAY *&readmaskp;
SRV_MASK_ARRAY *writemaskp;
SRV_MASK_ARRAY *exceptmaskp;
CS_INT          waitflag;
```

**パラメータ**

*nfds*

チェックするファイル記述子の最大番号です。

*&readmaskp*

読み込みが使用可能であることをチェックするためにファイル記述子のマスクで初期化された SRV\_MASK\_ARRAY 構造体へのポインタです。

*writemaskp*

書き込みが使用可能であることをチェックするためにファイル記述子のマスクで初期化された SRV\_MASK\_ARRAY 構造体へのポインタです。

*exceptmaskp*

例外をチェックするためにファイル記述子のマスクで初期化された SRV\_MASK\_ARRAY 構造体へのポインタです。

*waitflag*

任意のファイル記述子が希望の動作に対して使用可能となるまで、スレッドを保留しておかなければならないかどうかを示す CS\_INT です。waitflag の有効値については、次の「使用法」の項を参照してください。

**戻り値**

示された任意のオペレーションに対して使用可能であるファイル記述子の総数を返します。エラーが生じた場合には、-1 が返されます。

**表 3-116 : 戻り値 (srv\_select)**

戻り値	意味
整数	示されたどのオペレーションに対しても使用可能であるファイル記述子の総数。
-1	ルーチンが失敗。

**例**

```
#include <ospublic.h>
```

```

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_select PROTOTYPE((
CS_INT          readfd
));
/*
** EX_SRV_SELECT
**
** Example routine to illustrate the use of srv_select.
**
** Arguments:
** readfd - fd to be checked if it is &ready for a read **
**          operation.
**
** Returns:
** CS_SUCCEEDED      If readfd is &ready for a read operation.
** CS_FAIL           If readfd is not &ready for a read operation.
*/
CS_RETCODE      ex_srv_select(readfd)
CS_INT          readfd;
{
    SRV_MASK_ARRAY    &readmask;
    CS_BOOL           &ready;

    /* Initialization.*/
    (CS_VOID)srv_mask(CS_ZERO, &&readmask, (CS_INT)0, (CS_BOOL
        *)NULL);
    &ready = CS_FALSE;

    /* Set readfd in the mask.*/
    (CS_VOID)srv_mask(CS_SET, &&readmask, readfd, (CS_BOOL
        *)NULL);

    /*
    ** Check whether the descriptor is &ready for a read
    ** operation.If it is not, return.
    */
    if (srv_select(readfd+1, &&readmask, (SRV_MASK_ARRAY *)NULL,
        (SRV_MASK_ARRAY *)NULL, SRV_M_NOWAIT) <= 0 )
        return(CS_FAIL);

    /*
    ** A file descriptor is &ready for a read operation.
    */
}

```

```
(CS_VOID)srv_mask(CS_GET, &&readmask, readfd, &&ready);
return ((&ready) ?CS_SUCCEED :CS_FAIL);
```

```
}
```

#### 使用法

- I/O を要求せずに、ネットワーク I/O オペレーションをファイル記述子で行えるかどうかを調べたい場合には、`srv_select` を使用します。
- `Open Server` は、ファイル記述子の使用可能状況を調べるときに使うグローバル・マスクに、指定されたファイル記述子を含めます。
- `SRV_MASK_ARRAY` の定義は、次のとおりです。

```
#define SRV_MASK_SIZE                (CS_INT)40
#define SRV_MAXMASK_LENGTH
(CS_INT)(SRV_MASK_SIZE*CS_BITS_PER_LONG)
typedef struct          srv_mask_array
{
    long          mask_bits[SRV_MASK_SIZE];
} SRV_MASK_ARRAY;
```

`SRV_MASK_SIZE` は `SRV_MASK_ARRAY` の要素の数を示し、`SRV_MAXMASK_LENGTH` は `SRV_MASK_ARRAY` を使って表現することが可能なファイル記述子の最大数を示します。

- 外部ファイル記述子を使用する `Open Server` アプリケーションは、規則に従ってそれらをクローズしなければなりません。アプリケーション・スレッドは、外部ファイル記述子をクローズする前に、未処理の `srv_select` 呼び出しが完了するのを待たなければなりません。待たなかった場合、`Open Server` は終了します。
- [表 3-117](#) に、`waitflag` の有効値を示します。

表 3-117 : waitflag の値 (srv\_select)

値	意味
SRV_M_WAIT	スレッドは休止しており、マスクに表現されているファイル記述子のどれかが指定されたオペレーションのために使用可能になった場合に、ウェイクアップする。リターン・ステータスは、対象となるオペレーションに対して使用可能なファイル記述子があるかどうかを示す。
SRV_M_NOWAIT	ルーチンは、次のネットワーク・チェックの直後に戻る。リターン・ステータスは、対象となるオペレーションに対して使用可能なファイル記述子があるかどうかを示す。

- アプリケーションは、`srv_select` を使用して、ファイル記述子を調べてただちに戻るか、またはファイル記述子のどれかが準備ができるまでは戻らないようにするかを指定できます。
- `srv_select` は、`SRV_START` または `SRV_ATTENTION` ハンドラでは使用できません。

参照

[srv\\_mask](#)

## srv\_send\_ctlinfo

- 説明** Client-Library に制御メッセージを送信します。
- 構文** `CS_RETCODE srv_send_ctlinfo(SRV_PROC *srvproc, CS_INT ctl_type, SRV_CTL_MIGRATE ctl_type, CS_INT paramcnt, SRV_CTLITEM *param)`
- パラメータ**
- srvproc*  
内部スレッド制御構造体へのポインタです。
- ctl\_type*  
送信する制御メッセージのタイプです。
- paramcnt*  
*param* 配列の要素の数です。
- param*  
ライブラリ制御メッセージのパラメータです。
- 使用法**
- *ctl\_type* には、次の値を指定します。

- **SRV\_CTL\_MIGRATE**

マイグレーション要求をクライアントに送信するか、前のマイグレーション要求を取り消します。**SRV\_CTL\_MIGRATE** を使用できるのは、クライアントがマイグレーションをサポートし、初めてセッションに接続するときにセッション ID を受け取った場合だけです。

「**SRV\_CTL\_MIGRATE**」(43 ページ) を参照してください。

- **SRV\_CTL\_LOGINREDIRECT**

接続ハンドラ使用中のみ有効です。この値を使用すると、**SRV\_T\_REDIRECT** が true である **SRV\_PROC** は、渡されたサーバ・アドレスを使用してログインを再開するようクライアントに指示します。

- **SRV\_CTL\_HAUPDATE**

**srv\_sendinfo** が有効な場合はいつでも有効です。このメッセージがサーバからクライアントに送信されると、クライアントは現在の **HA** フェールオーバー・ターゲット情報を、*param* で表されたサーバ接続情報に置き換えます。

- *param* には次のフィールドがあります。

- **SRV\_CTLTYPES** *srv\_ctlitemtype* – *srv\_ctlitemtype* はパラメータの型を示します。次の型を使用できます。

- **SRV\_CT\_SERVERNAME** – *srv\_ctlptr* が、アドレスを検索する対象のサーバの名前が含まれている **CS\_CHAR** 文字列を指すことを示します。
- **SRV\_CT\_TRANADDR** – *srv\_ctlptr* が、サーバ・アドレス情報が含まれている **CS\_TRANADDR** 構造体を指すことを示します。
- **SRV\_CT\_ADDRSTR** – *srv\_ctlptr* が、**srv\_getserverbyname** によってフォーマットされた文字列を指すことを示します。
- **SRV\_CT\_OPTION** – *srv\_ctlptr* が、このメッセージに使用するオプション・セットが含まれている **CS\_INT** ビットマスクを指すことを示します。



- `CS_INT` *srv\_ctllength* — 変数サイズ・パラメータの長さです。*srv\_ctlitectype* が `SRV_CT_SERVERNAME` または `SRV_CT_ADDRSTR` である場合、*srv\_ctlptr* によって指された文字列の長さまたは `CS_NULLTERM` になります。*srv\_ctlitectype* が `SRV_CT_TRANADDR` である場合、*srv\_ctlptr* によって指された `CS_TRANADDR` 配列内の要素の数になります。
- `void *srv_ctlptr` — *srv\_ctlptr* は実際のパラメータ・データを指します。

参照

[srv\\_freeserveraddrs](#), [srv\\_getserverbyname](#)

## srv\_send\_data

説明

`srv_send_data` によって、Open Server アプリケーションでは、複数のカラムがあるローをクライアントに転送できます。そのため、Open Server アプリケーションでは、テキスト、イメージ、および XML のデータをまとまりとして送信でき、メモリが必要以上に消費されるのを防ぐことができます。

構文

```
CS_RETCODE srv_send_data(spp, column, buf, buflen)
SRV_PROC   *spp;
CS_INT     *column;
CS_BYTE    *buf;
CS_INT     buflen;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*column*

ロー・セットのカラムの数です。

*buf*

クライアントに送るデータを含むバッファへのポインタです。これは、セクションのサイズを決めます。

*buflen*

*\*buf* バッファの長さです。

戻り値

**表 3-118 : 戻り値 (srv\_send\_data)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ctpublic.h>
#include <ospublic.h>
/*
** Local Prototype.
*/
CS_RETCODE ex_srv_send_data PROTOTYPE((
SRV_PROC *spp,
CS_COMMAND *cmd,
CS_INT cols
));
#define MAX_BULK 51200

/*
** EX_SRV_SEND_DATA

** Example routine to demonstrate how to write columns
** of data in a row set to a client using srv_send_data.
** This routine will send all the columns of data read
** from a server back to the client.
** Arguments:
** spp          - A pointer to an internal thread control structure.
** cmd          - The command handle for the command that is returning
**               text data.
** cols         - The number of columns in a row set.
** Returns:
** CS_SUCCEED  - Result set sent successfully to client.
** CS_FAIL     - An error was detected.
*/
CS_RETCODE ex_srv_send_data(spp, cmd, cols)
SRV_PROC *spp;
CS_COMMAND *cmd;
CS_INT cols;
{
    CS_INT *len;          /* Length of column data.*/
    CS_INT *outlen;      /* Number of bytes received.*/
    CS_BYTE **data;      /* Column data.*/
    CS_BYTE buf[MAX_BULK]; /* Buffer for text data.*/
    CS_BOOL ok;          /* Error control flag.*/
    CS_INT i;
    CS_INT ret;
```

```
/* Initialization.*/
ok = CS_TRUE;

/*
** Transfer a row.
*/
for (i = 0; i < cols; i++)
{
    if ((fmt[i].datatype != CS_TEXT_TYPE) &&
        (fmt[i].datatype != CS_IMAGE_TYPE))
    {
        /*
        ** Transfer a non TEXT/IMAGE column.
        */

        /*
        ** Read the data of a non-text/image column
        ** from the server.
        */
        ret = ct_get_data(cmd, i+1, data[i],
            len[i], &outlen[i]);
        if ((ret != CS_SUCCEED) && (ret != CS_END_DATA)
            && (ret != CS_END_ITEM))
        {
            ok = CS_FALSE;
            break;
        }

        /*
        ** Write the data of a non-text/image column
        ** to client.
        */
        if (ret = srv_send_data(srvproc, i+1, NULL, 0)
            != CS_SUCCEED)
        {
            ok = CS_FALSE;
            break;
        }
    }
    else
    {
        /*
        ** Transfer a TEXT/IMAGE column in small trunks.
        */
    }
}
```

```

    /*
    ** Read a chunk of data of a text/image column
    ** from the server.
    */
    while ((ret = ct_get_data(cmd, i+1, buf, MAX_BULK, &len[i]))
           == CS_SUCCEED)
    {
        /*
        ** Write the chunk of data to client.
        */
        if (ret = srv_send_data(srvproc, i+1, buf, len[i])
            != CS_SUCCEED)
        {
            ok = CS_FALSE;
            break;
        }
    }
}

switch(ret)
{
    case CS_SUCCEED:
        /* The routine completed successfully.*/
    case CS_END_ITEM:
        /* Reached the end of this item's value.*/
    case CS_END_DATA:
        /* Reached the end of this row's data.*/
        break;
    case CS_FAIL:
        /* The routine failed.*/
    case CS_CANCELED:
        /* The get data operation was cancelled.*/
    case CS_PENDING:
        /* Asynchronous network I/O is in effect.*/
    case CS_BUSY:
        /* An asynchronous operation is pending.*/
    default:
        ok = CS_FALSE;
}
return (ok ? CS_SUCCEED : CS_FAIL);
}

```

使用法

- `srv_send_data` は、ロー・セットのカラムのデータをカラムごとにクライアントに送信します。

- テキスト・データ、イメージ・データ、または XML データを含んでいるカラムを送信するとき、Open Server アプリケーションが `srv_text_info` を呼び出してから `srv_send_data` を呼び出す必要があります。そうすることで、データ・ストリームが、送信されるデータの全体の長さに正しく設定されます。その後、アプリケーションがデータをまとめて送信するための `srv_send_data` を呼び出し、送信されるデータがなくなるまでこのルーチンの呼び出しが続行されます。
- Open Server アプリケーションでは、`srv_bind` と `srv_xferdata` を使用して、テキスト・データ、イメージ・データ、および XML データをクライアントに送信できます。ただし、これらのルーチンでは、すべてのデータ・カラムを一度に送信する必要があります。`srv_send_data` を使用すると、アプリケーションではテキスト・データとイメージ・データをまとめて送信できます。
- `srv_send_data` は一度に1つのカラムのデータを読み込んで送信するため、ロー・セットの最初のカラムと一緒にロー全体のデータ・フォーマットをクライアントに送信する必要があります。カラムを読み込む前のオブジェクト名などの固定 I/O フィールドを取得するには、`ct_data_info()` を呼び出します。`text` データや、`text` データの長さを指すポインタなど、I/O 記述子内で変更可能なフィールドは、カラムを読み込んだ後でのみ取得できることに注意してください。
- Open Server アプリケーションでは、文字セット変換を除き、テキスト・データ・ストリーム、イメージ・データ・ストリーム、および XML データ・ストリームを同一に扱います。これらの変換は、テキスト・データに対してのみ行われます。

## 参照

『Open Server 15.0 Server Library/C リファレンス・マニュアル』の関連する `srv_bind`、`srv_get_text`、`srv_text_info`、`srv_xferdata`、`srv_get_data`、`srv_send_text` ルーチン。

## srv\_send\_text

## 説明

連続したデータとして、`text` または `image` のデータ・ストリームをクライアントに送ります。

## 構文

```
CS_RETURN_CODE srv_send_text(spp, bp, buflen)
```

```
SRV_PROC      *spp;
CS_BYTE       *bp;
CS_INT        buflen;
```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*bp*  
クライアントに送るデータを含むバッファへのポインタです。これは、セクションのサイズを決めます。

*buflen*  
*\*bp* バッファのサイズです。

戻り値

**表 3-119: 戻り値 (srv\_send\_text)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ctpublic.h>
#include      <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_send_text PROTOTYPE((
SRV_PROC      *spp,
CS_COMMAND   *cmd
));

/*
** EX_SRV_SEND_TEXT
**
** Example routine to demonstrate how to write text to a client
** using srv_send_text. This routine will send all the text
** read from a server back to the client.
**
** Arguments:
**     spp      A pointer to an internal thread control structure.
**     cmd     The command handle for the command that is returning
**             text data.
**
** Returns:
**     CS_SUCCEED  Result set sent successfully to client.
**     CS_FAIL    An error was detected.
```

```

*/
CS_RETCODE      ex_srv_send_text(spp, cmd)
SRV_PROC        *spp;
CS_COMMAND      *cmd;
{
    CS_BOOL      ok;          /* Error control flag.*/
    CS_INT       ret;        /* ct_fetch return value.*/
    CS_INT       len_read;   /* Amount of data read.*/
    CS_BYTE      data[1024]; /* Buffer for text data.*/

    /* Initialization.*/
    ok = CS_TRUE;

    /* Read the text from the server.*/
    while ((ret = ct_get_data(cmd, 1, data, CS_SIZEOF(data),
        &len_read))
        == CS_SUCCEED)
    {
        /* Write text to client a chunk at a time */
        if (srv_send_text(spp, data, len_read) != CS_SUCCEED)
        {
            ok = CS_FALSE;
            break;
        }
    }

    switch(ret)
    {
    case CS_SUCCEED: /* The routine completed successfully.*/
    case CS_END_ITEM: /* Reached the end of this item's value.*/
    case CS_END_DATA: /* Reached the end of this item's value.*/
        break;
    case CS_FAIL: /* The routine failed.*/
    case CS_CANCELED: /* The get data operation was cancelled.*/
    case CS_PENDING: /* Asynchronous network I/O is in effect.*/
    case CS_BUSY: /* An asynchronous operation is pending.*/
    default:
        ok = CS_FALSE;
    }

    return (ok ? CS_SUCCEED : CS_FAIL);
}

```

#### 使用法

- `srv_send_text` は、クライアントに `text` や `image` データのカラムをひとつだけ送るのに使用します。

- Open Server アプリケーションは、データの全長を送るために、データ・ストリームに対して `srv_send_text` を最初に呼び出す前に、`srv_text_info` を呼び出す必要があります。それから、アプリケーションはまとまりを送るために `srv_send_text` を呼び出します。`srv_send_text` は、まとまりの数だけ呼び出します。
- クライアントから送られるアイテムは、あらかじめ `srv_descfmt` を使って記述されたものでなければなりません。
- また、Open Server アプリケーションは `srv_bind` と `srv_xferdata` を使って、`text` データと `image` データをクライアントに書き込むことができます。`srv_send_text` を使用するとデータを連続して送ることができますが、標準の `srv_bind/srv_xferdata` 方式ではカラムのすべてのデータを一度に送る必要があります。
- `srv_send_text` で送られたカラムは、`text` または `image` 型でなければなりません。
- `text` データだけに対して行われる文字セット変換を除き、Open Server は `text` と `image` のデータ・ストリームを同一に扱います。

---

**警告！** ローに 1 つのカラムしかなく、そのカラムが `text` または `image` データを含む場合にのみ、Open Server アプリケーションは `srv_send_text` を使用してローを送ることができます。

---

参照

[srv\\_bind](#), [srv\\_descfmt](#), [srv\\_get\\_text](#), [srv\\_text\\_info](#), [srv\\_xferdata](#), 「[text と image](#)」 (225 ページ)

## srv\_senddone

**説明** クライアントに結果完了メッセージを送るか、結果の一部をフラッシュします。

**構文** `CS_RETCODE srv_senddone(spp, status, transtate, count)`

```
SRV_PROC      *spp;
CS_INT        status;
CS_INT        transtate;
CS_INT        count;
```

**パラメータ**

*spp*  
内部スレッド制御構造体へのポインタです。



*status*

1つ以上のフラグの論理和から構成される2バイトのビットマスクです。表 3-120 に各フラグを示します。

表 3-120 : *status* の値 (*srv\_senddone*)

ステータス	説明
SRV_DONE_FINAL	結果の現在のセットは、結果の最終セット。
SRV_DONE_MORE	結果の現在のセットは、結果の最終セットではない。
SRV_DONE_COUNT	カウント・パラメータは有効カウントを含んでいる。
SRV_DONE_ERROR	現在のクライアント・コマンドでエラーが発生した。
SRV_DONE_FLUSH	現在の結果セットは、完全なパケットを待たずにクライアントに送られる。

*transtate*

トランザクションの現在のステータスです。表 3-121 に、*transtate* の有効値を示します。

表 3-121 : *transtate* の値 (*srv\_senddone*)

トランザクションのステータス	説明
CS_TRAN_UNDEFINED	現在トランザクション中ではない。
CS_TRAN_COMPLETED	現在のトランザクションは正常に終了した。
CS_TRAN_FAIL	現在のトランザクションは失敗した。
CS_TRAN_IN_PROGRESS	現在トランザクションにある。
CS_TRAN_STMT_FAIL	現在のトランザクション文は失敗した。

*count*

結果の現在のセットのカウントを含む4バイトのフィールドです。*status* フィールドに *SRV\_DONE\_COUNT* フラグが設定してある場合、カウントは有効です。

## 戻り値

表 3-122: 戻り値 (*srv\_senddone*)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_senddone PROTOTYPE((
SRV_PROC *spp
```

```

));

/*
** Constants and data definitions.
*/
#define    NUMROWS        3
#define    MAXROWDATA    6

CS_STATIC CS_CHAR    *row_data[NUMROWS] = {
    "Larry",
    "Curly",
    "Moe"
};

/*
** EX_SRV_SENDDONE
**
** Example routine illustrating the use of srv_senddone. This
** routine will send a set of results to the client
** application, and then send the results completion message.
**
** Arguments:
**     spp        A pointer to an internal thread control structure.
**
** Returns:
**     CS_SUCCEED    Results set sent successfully to client.
**     CS_FAIL        An error was detected.
*/
CS_RETCODE    ex_srv_senddone(spp)
SRV_PROC        *spp;
{
    CS_DATAFMT    fmt;
    CS_INT        row_len;
    CS_INT        idx;

    /*
    ** Describe the format of the row data, with the single
    ** dummy column.
    */
    srv_bzero((CS_VOID *)&fmt, (CS_INT)sizeof(fmt));
    fmt.datatype = CS_CHAR_TYPE;
    fmt.maxlength = MAXROWDATA;

    if (srv_descfmt(spp, (CS_INT)CS_SET, (CS_INT)SRV_ROWDATA,
        (CS_INT)1, &fmt) != CS_SUCCEED)
    {

```

```

        (CS_VOID)srv_senddone(spp,
            (CS_INT)(SRV_DONE_FINAL | SRV_DONE_ERROR),
            (CS_INT)CS_TRAN_FAIL, (CS_INT)0);
    return(CS_FAIL);
}

for (idx = 0; idx < NUMROWS; ++idx)
{
    /*
    ** Bind the row_data array element.
    */
    row_len = (CS_INT)strlen(row_data[idx]);
    if (srv_bind(spp, (CS_INT)CS_SET, (CS_INT)SRV_ROWDATA,
        (CS_INT)1, &fmt, (CS_BYTE *) (row_data[idx]),
        &row_len, (CS_SMALLINT *)NULL) != CS_SUCCEEDED)
    {
        /* Communicate failure, and number of rows sent.*/
        (CS_VOID)srv_senddone(spp,
            (CS_INT)(SRV_DONE_FINAL |
                SRV_DONE_ERROR | SRV_DONE_COUNT),
            (CS_INT)CS_TRAN_FAIL, (CS_INT)idx);
        return(CS_FAIL);
    }

    /*
    ** Transfer the row data.
    */
    if (srv_xferdata(spp, (CS_INT)CS_SET, SRV_ROWDATA)
        != CS_SUCCEEDED)
    {
        /* Communicate failure, and number of rows sent.*/
        (CS_VOID)srv_senddone(spp,
            (CS_INT)(SRV_DONE_FINAL |
                SRV_DONE_ERROR | SRV_DONE_COUNT),
            (CS_INT)CS_TRAN_FAIL, (CS_INT)idx);
        return(CS_FAIL);
    }
}

/* Send a status value.*/
if (srv_sendstatus(spp, (CS_INT)0) != CS_SUCCEEDED)
{
    /* Communicate failure, and number of rows sent.*/
    (CS_VOID)srv_senddone(spp,
        (CS_INT)(SRV_DONE_FINAL |

```

```

        SRV_DONE_ERROR | SRV_DONE_COUNT),
        (CS_INT)CS_TRAN_FAIL, (CS_INT)NUMROWS);
    return(CS_FAIL);
}

/* Send the final DONE message, with the row count.*/
if (srv_senddone(spp, (CS_INT)(SRV_DONE_FINAL |
    SRV_DONE_COUNT),
    (CS_INT)CS_TRAN_COMPLETED,
    (CS_INT)NUMROWS) != CS_SUCCEEDED)
{
    /* Communicate failure, and number of rows sent.*/
    (CS_VOID)srv_senddone(spp,
        (CS_INT)(SRV_DONE_FINAL |
            SRV_DONE_ERROR | SRV_DONE_COUNT),
        (CS_INT)CS_TRAN_FAIL, (CS_INT)NUMROWS);
    return(CS_FAIL);
}
return (CS_SUCCEEDED);
}

```

#### 使用法

- `srv_senddone` は、結果の現在のセットが完了したというメッセージをクライアントに送信します。クライアント要求は、サーバに複数のコマンドを実行させ、複数の結果セットを返させることもできます。各結果セットについて、`srv_senddone` で完了メッセージが返されなければなりません。
- 現在の結果がクライアント・コマンド・バッチの最後の結果セットでない場合には、**Open Server** は *status* マスクの **SRV\_DONE\_MORE** フィールドを設定しなければなりません。設定しない場合には、**Open Server** アプリケーションは現在のコマンド・バッチに対する結果が、これ以上ないことを示すために、**SRV\_DONE\_FINAL** を *status* フィールドに設定しなければなりません。
- *count* フィールドは、特定のコマンドにより影響を受けたローがいくつあるのかを示します。*count* に実際のカウントがある場合には、**SRV\_DONE\_COUNT** ビットが *status* フィールドに設定されていなくてはなりません。こうすることにより、クライアントは 0 の実際のカウントと未使用の *count* フィールドの区別が可能になります。

- `SRV_CONNECT` ハンドラがクライアント・ログインを拒否した場合には、Open Server アプリケーションは `status` パラメータを `SRV_DONE_ERROR` フラグに設定して、`srv_senddone` を呼び出す必要があります。その場合、`SRV_CONNECT` ハンドラは、`srv_senddone` を使って、クライアントに `DONE` パケットを送信しなければなりません。いずれにしても、`SRV_CONNECT` ハンドラが戻り、`SRV_DONE_FINAL status` フラグが設定される前に、`srv_senddone` を一度だけ呼び出さなければなりません。
- 書き込み中にネットワーク・バッファが満杯になった場合、Open Server はその内容をフラッシュします。ネットワーク・バッファがどれだけの内容を含んでいるかには関係なく、`status` を、`SRV_DONE_FINAL` または `SRV_DONE_FLUSH` に設定した状態で、`srv_senddone` を発行することによって、ネットワーク・バッファのフラッシュを引き起こします。`SRV_DONE_FLUSH` は `SRV_DONE_MORE` の指定に関わらず、設定できます。
- `status` を `SRV_DONE_FLUSH` に設定すると、アプリケーションは長期間にわたって蓄積したクライアント側の結果をフラッシュできます。
- アプリケーションは、`SRV_CONNECTION` エラー・ハンドラ内では `status` 引数を `SRV_DONE_FLUSH` に設定することはできません。
- Open Server は、トランザクション管理を行いません。要求されたとおりに `transtate` 引数を使用して、クライアントに対して現在のトランザクション・ステータスを通知するのは、Open Server アプリケーションが行います。

---

**注意** `transtate` 引数は、Open Server 2.0 バージョンの `srv_senddone` の `info` 引数に代わるものです。現行のアプリケーションの `info` の値が 0 でない場合、現行のアプリケーションではランタイム・エラーが発生します。

---

参照

[srv\\_bind](#), [srv\\_descfmt](#), [srv\\_sendstatus](#), [srv\\_xferdata](#)

## srv\_sendinfo

説明

クライアントにエラー・メッセージを送ります。

構文

CS\_RETURNCODE `srv_sendinfo`(spp, errmsgp, transtate)

```
SRV_PROC      *spp;
CS_SERVERMSG  *errmsgp;
CS_INT        transtate;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*errmsgp*

クライアントに送られるエラー・メッセージ情報を含む  
**CS\_SERVERMSG** 構造体へのポインタです。「[CS\\_SERVERMSG 構造体](#)」(64 ページ)を参照してください。

*transtate*

トランザクションの現在のステータスです。表 3-123 に、*transtate* の有効値を示します。

**表 3-123 : transtate の値 (srv\_sendinfo)**

トランザクションのステータス	説明
CS_TRAN_UNDEFINED	現在トランザクション中ではない。
CS_TRAN_COMPLETED	現在のトランザクションは正常に終了した。
CS_TRAN_FAIL	現在のトランザクションは失敗した。
CS_TRAN_IN_PROGRESS	現在トランザクションにある。
CS_TRAN_STMT_FAIL	現在のトランザクション文は失敗した。

戻り値

**表 3-124 : 戻り値 (srv\_sendinfo)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local Prototype.
*/

CS_RETCODE      ex_srv_sendinfo  PROTOTYPE((
SRV_PROC        *sp,
CS_CHAR         *msg,
CS_INT          msglen,
CS_INT          msgnum
));

/*
** EX_SRV_SENDINFO
```

```
**
**      This routine demonstrates how to use srv_sendinfo to send
**      an error message to a client.
**
** Arguments:
**      sp          A pointer to an internal thread control
**      structure.
**      msg         The message text to send.
**      msglen     The length of the message text to send.
**      msgnum     The message number to send.
**
** Returns
**      CS_SUCCEED   If the message is sent.
**      CS_FAIL     If an error occurred.
**
*/
CS_RETCODE    ex_srv_sendinfo(sp, msg, msglen, msgnum)
SRV_PROC      *sp;
CS_CHAR      *msg;
CS_INT       msglen;
CS_INT       msgnum;
{
    CS_SERVERMSG    &mrec;

    /*
    ** Initialization.
    */
    srv_bzero(&&mrec, sizeof(CS_SERVERMSG));

    /*
    ** First, determine if the message string will fit
    ** in the message structure. If not, truncate it.
    */
    if( msglen > CS_MAX_MSG )
    {
        msglen = CS_MAX_MSG;
    }

    /*
    ** Now copy the message string over.
    */
    srv_bmove(msg, &mrec.text, msglen);
    &mrec.textlen = msglen;

    /*
    ** Set the message number we want to send.
    */
}
```

```

&mrec.msgnumber = msgnum;

/* Set the message status so that &mrec.text contains
** the entire message
*/
&mrec.status = CS_FIRST_CHUNK | CS_LAST_CHUNK;

/*
** Now we're &ready to send the message.
*/
if( srv_sendinfo(sp, &mrec, CS_TRAN_UNDEFINED) == CS_FAIL )
{
    /*
    ** An error was al&ready raised.
    */
    return CS_FAIL;
}

/*
** All done.
*/
return CS_SUCCEED;
}

```

#### 使用法

- `srv_sendinfo` は、クライアントにエラー・メッセージを送信します。送信された各メッセージで一度ずつ呼び出さなければなりません。
- アプリケーションは、結果ローの送信の前後にかかわらず、`srv_sendinfo` を呼び出すことができます。ただし、アプリケーションは、`srv_descfmt` に連続して呼び出している間や、`srv_descfmt` と `srv_xferdata` の呼び出しの間に、`srv_sendinfo` を呼び出すことはできません。
- Open Server アプリケーションが、エラー・メッセージに関するパラメータ・データを送りたい場合には、`CS_SERVERMSG` 構造体の `status` フィールドを、`CS_HASEED` に設定します。アプリケーションは `srv_sendinfo` を呼び出した直後に、他の結果を送ったり、`srv_senddone` を呼び出す前に、エラー・パラメータを記述、バインド、送信しなければなりません。アプリケーションは、`type` 引数を `SRV_ERRORDATA` に設定して、`srv_descfmt`、`srv_bind`、`srv_xferdata` を呼び出さなければなりません。



- アプリケーションが、CS\_SERVERMSG 構造体の *status* フィールドを CS\_HASEED に設定して *srv\_sendinfo* を呼び出しても、エラー・パラメータを送らなかった場合には、アプリケーションが *srv\_senddone* を呼び出した時点で、致命的なエラーとなります。
- アプリケーションが CS\_SERVERMSG 構造体の *status* フィールドを CS\_HASEED に設定した状態で *srv\_sendinfo* を呼び出すと、Open Server は、CS\_RES\_NOEED 応答機能が設定されていないことを確認します。設定されている場合には、Open Server は、エラーを出します。それ以降、エラー・パラメータを記述するために *srv\_descfmt* を呼び出したものも、エラーを引き起こします。
- 「クライアント・コマンド・エラー」(38 ページ) を参照してください。
- 「CS\_SERVERMSG 構造体」(64 ページ) も参照してください。

## 参照

*srv\_bind*, *srv\_descfmt*, *srv\_senddone*, *srv\_xferdata*, 「クライアント・コマンド・エラー」(38 ページ)

## srv\_sendpassthru

## 説明

クライアントにプロトコル・パケットを送ります。

## 構文

```
CS_RETCODE srv_sendpassthru(spp, send_bufp, infop)
```

```
SRV_PROC   *spp;
CS_BYTE    *send_bufp;
CS_INT     *infop;
```

## パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*send\_bufp*

プロトコル・パケットが格納されているバッファへのポインタです。

*infop*

*srv\_sendpassthru* が CS\_FAIL を返した場合に SRV\_I\_UNKNOWN に設定される CS\_INT へのポインタです。表 3-125 は、このルーチンが CS\_SUCCEED を返す場合に *\*infop* に返される可能性のある値を示します。

**表 3-125 : CS\_SUCCEED の値 (srv\_sendpassthru)**

値	説明
SRV_I_PASSTHRU_MORE	プロトコル・パケットの送信は成功し、メッセージ・パケットの終了ではない。
SRV_I_PASSTHRU_EOM	メッセージ・プロトコル・パケットの最後の送信は成功した。

戻り値

**表 3-126 : 戻り値 (srv\_sendpassthru)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <stdio.h>
#include      <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_sendpassthru PROTOTYPE((
SRV_PROC        *spp
));

/*
** EX_SRV_SENDFASSTHRU
**
** Example routine to send a protocol packet to a client.
**
** Arguments:
**     spp      A pointer to an internal thread control structure.
**
** Returns:
**
**     CS_SUCCEED
**     CS_FAIL
**
*/
CS_RETCODE      ex_srv_sendpassthru(spp)
SRV_PROC        *spp;
{
    CS_BYTE      sendbuf[20];
    CS_INT       info;

    strcpy(sendbuf, "Here's what to send");
```

```

if (srv_sendpassthru(spp, sendbuf, &info) == CS_FAIL)
{
    return(CS_FAIL);
}
else
{
    if (info == SRV_I_PASSTHRU_MORE)
    {
        printf("more to come...¥n");
        return (CS_SUCCEED);
    }
    else if (info == SRV_I_PASSTHRU_EOM)
    {
        printf("That's all.¥n");
        return (CS_SUCCEED);
    }
    else
    {
        printf("Unknown flag returned.¥n");
        return(CS_FAIL);
    }
}
}

```

#### 使用法

- `srv_sendpassthru` はクライアント・プログラムまたは Adaptive Server Enterprise から受けたプロトコル・パケットを、内容を判断せずにそのまま転送します。
- `srv_sendpassthru` は、プロトコル・ヘッダ・フィールドのバイト順を設定します。
- 一度 `srv_sendpassthru` が呼び出されると、それを呼び出したスレッドはパススルー・モードになります。パススルー・モードは、`SRV_PASSTHRU_EOM` が返されると終了します。
- イベント・ハンドラがパススルー・モード状態の場合は、他の Server-Library ルーチンを呼び出すことはできません。
- パススルー・モードを使用するには、クライアントの `SRV_CONNECT` ハンドラは、`srv_getloginfo`、`ct_setloginfo`、`ct_getloginfo`、`srv_setloginfo` を呼び出すことによって、クライアントとリモート・サーバがプロトコルをネゴシエーションできるようにしなければなりません。これによって、異なるプラットフォーム上で動作しているクライアントおよびリモート・サーバは、必要なデータ変換を行うことができます。

- `srv_sendpassthru` は、`SRV_CONNECT`、`SRV_DISCONNECT`、`SRV_START`、`SRV_STOP`、`SRV_URGDISCONNECT`、`SRV_ATTENTION` を除くすべてのイベント・ハンドラで使用できません。

参照 [srv\\_getloginfo](#), [srv\\_recvpassthru](#), [srv\\_setloginfo](#)

## srv\_sendstatus

説明 クライアントにステータス値を送信します。

構文 `CS_RETCODE srv_sendstatus(spp, value)`

```
SRV_PROC      *spp;
CS_INT        value;
```

パラメータ *spp*  
内部スレッド制御構造体へのポインタです。

*value*  
要求のステータスです。規則により、0 は要求が正常に終了したことを示します。

戻り値 **表 3-127 : 戻り値 (srv\_sendstatus)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local prototype.
*/
CS_RETCODE      ex_srv_sendstatus PROTOTYPE((
SRV_PROC        *spp
));

/*
** EX_SRV_SENDSTATUS
**
** Example routine to send a status value to a client.
**
** Arguments:
```

```

**      spp      A pointer to an internal thread control structure.
**
** Returns:
**      CS_SUCCEEDED if we were able to send the status.
**      CS_FAIL if an error was detected.
**
**/
CS_RETCODE      ex_srv_sendstatus(spp)
SRV_PROC        *spp;
{
    CS_RETCODE      result;

    /*
    ** Send an OK status.
    **/
    result = srv_sendstatus(spp, (CS_INT)0);

    return (result);
}

```

#### 使用法

- `srv_sendstatus` は、クライアント要求に応答して、クライアントにリターン・ステータス値を送信します。要求を受信すると、ユーザ定義のイベント・ハンドラ・ルーチンが呼び出されます。要求に対する応答の一部は、ステータス値を返すためのものである場合があります。
- `srv_sendstatus` によって送られたステータス値は任意であり、アプリケーション指定のものです。これは `srv_senddone status` パラメータとは関連していません。
- すべてのロー (存在する場合) は、`srv_xferdata` でクライアントに送られた後、そして `srv_senddone` で完了ステータスが送られる前に、ステータス値を送ることができます。ステータス値を `srv_descfmt` および `srv_bind` の呼び出しと `srv_xferdata` の呼び出しの間に送信することはできません。
- 各結果セットに対して送信できるステータス値は 1 つだけです。

#### 参照

[srv\\_senddone](#)

## srv\_setcolutype

#### 説明

カラムと関連付けられるユーザ・データ型を指定します。

構文 CS\_RETCODE srv\_setcolutype(spp, column, utype)  
 SRV\_PROC \*spp;  
 CS\_INT column;  
 CS\_INT utype;

パラメータ *spp*  
 内部スレッド制御構造体へのポインタです。

*column*  
 ユーザ・データ型の関連対象となるカラムの番号です。先頭のカラムは1です。

*utype*  
 カラムに関連するユーザ定義のデータ型です。

戻り値 **表 3-128 : 戻り値 (srv\_setcolutype)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_setcolutype PROTOTYPE((
SRV_PROC *spp,
CS_INT column,
CS_INT utype
));

/*
** EX_SRV_SETCOLUTYPE
**
** Example routine to define the user datatype to be associated
** with a column using srv_setcolutype.
**
** Arguments:
** spp A pointer to an internal thread control structure.
** column The column number associated with the type.
** utype The type to be associated with the column.
**
** Returns:
**
** CS_SUCCEED The datatype was successfully associated with
```

```

    ** the column.
    **      CS_FAIL          An error was detected.
    */
CS_RETCODE      ex_srv_setcolutype(spp, column, utype)
SRV_PROC        *spp;
CS_INT          column;
CS_INT          utype;
{
    /*
    ** Associate the type with the column.
    */
    if (srv_setcolutype(spp, column, utype) != CS_SUCCEEDED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}

```

**使用法**

`srv_setcolutype` によって設定されたデータ型は、クライアント・アプリケーションが **DB-Library** 呼び出しの `dbcolutype` または **Client-Library** 呼び出しの `ct_describe` によって受け取るデータ型です。

## srv\_setcontrol

**説明**

カラムのユーザ制御またはフォーマット情報を記述します。

**構文**

```

CS_RETCODE      srv_setcontrol(spp, colnum, ctrlinfo,
                               ctrlrlen)
SRV_PROC        *spp;
CS_INT          colnum;
CS_BYTE         *ctrlinfo;
CS_INT          ctrlrlen;

```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*colnum*

制御情報が適用するカラムの番号です。ローの先頭カラムの番号は 1 です。

*ctrlinfo*

制御データへのポインタです。制御データの長さは、`ctrlrlen` パラメータで取得します。

*ctrl*len

制御データ長のバイト数です。各カラムは、最大で SRV\_MAXCHAR バイトの制御情報があります。

戻り値

**表 3-129: 戻り値 (srv\_setcontrol)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_setcontrol PROTOTYPE((
SRV_PROC        *spp
));

/*
** Constants.
*
#define          MAXROWDATA          20
#define          COLCONTROL          "Emp name:%s"

/*
** EX_SRV_SETCONTROL
**
** Example routine to describe format information for a column
** using srv_setcontrol. In this example, a simple character
** column contains an employee name.
**
** Arguments:
**     spp      A pointer to an internal thread control structure.
**
** Returns:
**     CS_SUCCEED      Control information successfully defined.
**     CS_FAIL        An error was detected.
*/
CS_RETCODE      ex_srv_setcontrol(spp)
SRV_PROC        *spp;
{
    CS_DATAFMT    fmt;

    /* Describe the format of the row data for the column.*/
```



```

    srv_bzero((CS_VOID *)&fmt, (CS_INT)sizeof(fmt));
    fmt.datatype = CS_CHAR_TYPE;
    fmt.maxlength = MAXROWDATA;

    if (srv_descfmt(spp, (CS_INT)CS_SET, (CS_INT)SRV_ROWDATA,
        (CS_INT)1, &fmt) != CS_SUCCEED)
    {
        return(CS_FAIL);
    }

    /* Define the control information for the column.*/
    if (srv_setcontrol(spp, (CS_INT)1, (CS_BYTE *)COLCONTROL,
        (CS_INT)strlen(COLCONTROL)) != CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

#### 使用法

- **Open Server** アプリケーションは、特定のカラムに関連するユーザ定義フォーマット情報を、`srv_setcontrol` を使用してクライアントに伝えます。たとえば、クライアントが、特定のカラムに合わせて、特定の文字列を送信することを希望しているような場合です。
- `srv_setcontrol` の呼び出しは、`srv_xferdata` の呼び出しの前か、`srv_descfmt` の呼び出しの後に行わなければなりません。他のコンテキストから呼び出すと、`CS_FAIL` を返します。
- 制御情報は、順序に関係なくカラムと関連付けることができます。唯一の条件は、カラムがあらかじめ `srv_descfmt` で定義されていないということなのです。
- ローのすべてのカラムについて、`srv_setcontrol` を呼び出す必要はありません。**Open Server** アプリケーションが、カラムの制御情報を設定しなければ、`null` 制御文字列が、そのカラムについて返されます。
- クライアントが、クライアント・オプションの切り替えを行う `CS_OPT_CONTROL` によって、具体的に制御情報を要求していない場合、アプリケーションは、このような情報を返さないようにする必要があります。

#### 参照

[srv\\_bind](#), [srv\\_descfmt](#), [srv\\_xferdata](#)

## srv\_setloginfo

**説明** リモート・サーバからクライアントにプロトコルのフォーマット情報を返します。

**構文** CS\_RETCODE srv\_setloginfo(spp, loginfop)

```
SRV_PROC      *spp;
CS_LOGINFO    *loginfop;
```

**パラメータ**

*spp*

内部スレッド制御構造体へのポインタです。

*loginfop*

ct\_getloginfo によって更新された CS\_LOGINFO 構造体へのポインタです。

**戻り値**

**表 3-130 : 戻り値 (srv\_setloginfo)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

**例**

```
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_setloginfo      PROTOTYPE((
SRV_PROC        *spp,
CS_LOGINFO      *loginfop
));

/*
** EX_SRV_SETLOGINFO
**
** Return protocol format information from a remote server to
** a client.
**
** Arguments:
**
** spp          A pointer to an internal thread control structure.
** loginfop     A pointer to a CS_LOGINFO structure that has been
**              updated by ct_getloginfo.
**
** Returns
```

```

**
**      CS_SUCCEED
**      CS_FAIL
**
*/
CS_RETCODE   ex_srv_setloginfo(spp, loginfop)
SRV_PROC     *spp;
CS_LOGININFO *loginfop;
{
    /* Check arguments.*/
    if (spp == (SRV_PROC *)NULL)
    {
        return(CS_FAIL);
    }
    return(srv_setloginfo(spp,loginfop));
}

```

#### 使用法

- クライアントおよびリモート Sybase サーバ間で、パケットの内容を判断せずに、プロトコル (TDS) パケットを渡すゲートウェイ・サーバ・アプリケーションでは、`srv_setloginfo` を使用します。
- クライアントが直接サーバに接続する場合は、2つのプログラムは、データの送受信に使用するプロトコル・フォーマットをネゴシエートします。ゲートウェイ・アプリケーションにおいて、プロトコル・パススルーを使用すると、**Open Server** は、クライアントとリモート・サーバ間において、プロトコル・パケットを転送します。
- `srv_setloginfo` は、クライアントとリモート・サーバ間の TDS フォーマットのネゴシエーションを可能にする 4つの呼び出しの 4番目です (そのうち2つは **CT-Library** 呼び出しです)。これらの呼び出しは、**SRV\_CONNECT** イベント・ハンドラにおいてのみ行えます。次にその呼び出しを示します。
  - a `srv_getloginfo` — `CS_LOGININFO` 構造体を割り付け、クライアント・スレッドからの TDS 情報を格納します。
  - b `ct_setloginfo` — 手順 1 で取得したプロトコル情報を使用して `CS_LOGININFO` 構造体を準備し、`ct_connect` を使用してリモート・サーバにログインします。
  - c `ct_getloginfo` — `CS_CONNECTION` 構造体から、新しく割り付けられた `CS_LOGININFO` 構造体にプロトコル・ログイン応答情報を転送します。
  - d `srv_setloginfo` — 手順 3 で取得したリモート・サーバの応答をクライアントに送信し、`CS_LOGININFO` 構造体を解放します。

参照

[srv\\_getlogininfo](#), [srv\\_recvpassthru](#), [srv\\_sendpassthru](#)

## srv\_setpri

説明

スレッドのスケジューリング優先順位を変更します。

構文

CS\_RETCODE `srv_setpri(spp, mode, priority_value)`

```
SRV_PROC      *spp;
CS_INT        mode;
CS_INT        priority_value;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*mode*

*priority\_value* が現在の優先順位を調節する場合には `SRV_C_DELTAPRI`、*priority\_value* が新しい優先順位である場合には `SRV_C_NEWPRI` となります。

*priority\_value*

*mode* が `SRV_C_NEWPRI` の場合には、*priority\_value* がスレッドの新しい優先順位です。*mode* が `SRV_C_DELTAPRI` の場合には、負の *priority\_value* は現在の優先順位からその絶対値を減らし、正の *priority\_value* は現在の優先順位を上げます。

戻り値

**表 3-131 : 戻り値 (srv\_setpri)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include      <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE   ex_srv_setpri PROTOTYPE((
SRV_PROC     *spp,
CS_INT       mode,
CS_INT       priority
));
```

```

/*
** EX_SRV_SETPRI
**
**     Example routine to change a thread' s scheduling priority.
**
** Arguments:
**     spp           A pointer to an internal thread control structure.
**     mode          Indicates whether a priority is relative or
**                   absolute.
**     priority      The change in priority value or the new
**                   priority value.
**
** Returns:
**
**     CS_SUCCEED
**     CS_FAIL
*/
CS_RETCODE   ex_srv_setpri(spp, mode, priority)
SRV_PROC     *spp;
CS_INT       mode;
CS_INT       priority;
{
    return(srv_setpri(spp, mode, priority));
}

```

#### 使用法

- クライアントが Open Server にログインした結果、あるいは `srv_createproc` や `srv_spawn` の呼び出しの結果としてスレッドが起動した場合には、優先順位は `SRV_C_DEFAULTPRI` になります。
- `srv_setpri` は、新しい値の設定、または現在の値を指定した値まで上昇/減少させることによって、優先順位を変更することができます。
- あるスレッドが、自分の優先順位よりも、他のスレッドの優先順位を高く設定した場合には、後者のスレッドはただちに優先順位が高くなるようスケジューラされます。それ以外の場合には、影響を受けたスレッドの新しい優先順位は、次にスケジューラが実行されるときから有効になります。
- スリープすることのないスレッドに他のスレッドよりも高い優先順位を持たせると、他のスレッドを実行する機会はまったくなくなります。
- 内部 Open Server スレッドは、`SRV_C_DEFAULTPRI` の優先順位で実行します。スレッドの優先順位を `SRV_C_DEFAULTPRI` 以上に設定した場合には、このような内部スレッドの実行を可能にするために、ときおりスリープさせる必要があります。

- 優先順位を `SRV_C_LOWPRIORITY` より低く設定したり、`SRV_C_MAXPRIORITY` より高く設定したりすると、エラーになります。
- `srv_setpri` は、`SRV_START` ハンドラでは使用できません。

参照

[srv\\_createproc](#), [srv\\_spawn](#)

## srv\_signal (UNIX のみ)

説明 シグナル・ハンドラをインストールします。

構文 `SRV_SIGNAL_FUNC srv_signal(sig, handler)`

```
CS_INT          sig;
SRV_SIGNAL_FUNC handler;
```

パラメータ

*sig*

ハンドラがインストールされる対象の UNIX シグナルの番号です。これは、*sgs/signal.h* で定義されます。

*handler*

*sig* が Open Server に対して配信されるときに、呼び出される関数へのポインタです。*handler* を `SIG_DFL` に設定すると、デフォルト・ハンドラがリストアされます。*handler* を `SIG_IGN` に設定すると、*sig* が無視されます。

戻り値

**表 3-132 : 戻り値 (srv\_signal)**

戻り値	意味
以前にインストールされたハンドラ関数へのポインタ	関数のロケーション。
null ポインタ	ルーチンが失敗。

例

```
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <ospublic.h>
/*
** Local Prototype.
*/
CS_STATIC CS_VOID ex_sigio_handler PROTOTYPE((
CS_INT          sig
));
```

```
CS_RETCODE      ex_srv_signal PROTOTYPE((
CS_INT          *uerrno
));

/*
** Static storage.
*/
CS_STATIC CS_INT io_events = 0;

/*
** EX_SRV_SIGNAL
**
** Example routine to install a UNIX signal handler for SIGIO,
** using srv_signal.
**
** Arguments:
** uerrno    A pointer to a user's error number indicator.
**
** Returns:
**
** CS_SUCCEED    Handler successfully installed.
** CS_FAIL      Handler not installed, UNIX global errno set.
*/
CS_RETCODE      ex_srv_signal(uerrno)
CS_INT          *uerrno;
{
    /*
    ** Install the handler.
    */
    (CS_VOID)srv_signal((int)SIGIO,
                        (SRV_SIGNAL_FUNC)ex_sigio_handler);

    /* Was there an error condition?*/
    if ((*uerrno = errno) != 0)
        return(CS_FAIL);

    return (CS_SUCCEED);
}

/*
** EX_SIGIO_HANDLER
**
** Example signal handler to count I/O events. It prints a
** message when the Open Server application has been up long
** enough to get 100,000 I/O events.
```

```

**
** Arguments:
**     sig  The signal number, always SIGIO.
**
** Returns:
**     Nothing.
*/
CS_STATIC CS_VOID    ex_sigio_handler(sig)
CS_INT             sig;
{
    if (io_events == 100000)
    {
        fprintf(stderr, "The server has been up a long
            time!!\n");
        io_events = 0;
    }
    else
    {
        io_events++;
    }
}

```

**使用法**

- Open Server は、SIGIO と SIGURG の UNIX シグナル・ハンドラをインストールします。これらのハンドラは、一度 Open Server が起動すると、常にアクティブの状態であればなりません。ハンドラがアクティブの状態でない場合には、Server-Library の I/O やアテンション処理ルーチンは機能に障害を生じるか、あるいは信頼性が損なわれます。

---

**警告！** sigvec(2) または signal(2) を使用して UNIX シグナル・ハンドラをインストールすると、予期できない結果を引き起こす場合があります。アプリケーションでは、srv\_signal を使用するようになしてください。

---

- Open Server は、アプリケーションがシグナル・ハンドラにある間、他のすべてのシグナルがブロックされることを保証します。
- signal に関する UNIX のマニュアルを参照してください。

## srv\_sleep

**説明**

現在実行中のスレッドを休止します。



```

構文      CS_RETCODE srv_sleep(sleepeventp, sleeplabelp,
                    sleepflags, infop, reserved1,
                    reserved2)

CS_VOID   *sleepeventp;
CS_CHAR   *sleeplabelp;
CS_INT    sleepflags;
CS_INT    *infop;
CS_VOID   *reserved1;
CS_VOID   *reserved2;

```

## パラメータ

*sleepeventp*

*srv\_wakeup* は、スレッドをウェイクアップするために使う汎用の void ポインタです。このポインタは、スレッドがスリープ中のオペレーティング・システム・イベントに対してユニークなものでなければなりません。たとえば、メッセージが別のスレッドに渡される場合、送信スレッドは、そのメッセージが処理されるまでスリープします。メッセージへのポインタは、受信スレッドが送信側をウェイクアップさせるために *srv\_wakeup* に渡せる有効な *sleepevent* です。

*sleeplabelp*

スレッドがスリープ中のイベントを識別する、null で終了する文字列へのポインタです。これは、スレッドがスリープしている理由を判別するのに役立ちます。アプリケーションは、Open Server システム・レジスタード・プロシージャの *sp\_ps* を使って、この情報を表示することができます。

*sleepflags*

このフラグの値は、スレッドがウェイクアップする状況を決定します。表 3-133 に、*sleepflags* の有効値を示します。

表 3-133 : *sleepflags* の値 (*srv\_sleep*)

値	説明
SRV_M_ATTNWAKE	スレッドがアテンションを受け取るとウェイクアップする。
SRV_M_NOATTNWAKE	アテンションは、スレッドをウェイクアップさせられない。

*infop*

CS\_INT を指すポインタです。表 3-134 は、*srv\_sleep* が CS\_FAIL を返す場合に *\*infop* に返される可能性がある値を示します。

**表 3-134 : infop の値 (srv\_sleep)**

値	説明
SRV_I_INTERRUPTED	スレッドは、 <code>srv_ucwakeup</code> によって無条件にウェイクアップさせられた。
SRV_I_UNKNOWN	その他のエラーが発生した。たとえば、スレッドはすでにスリープしている、または無効となっている。

*reserved1*

ミューテックスに対するプラットフォーム依存のハンドルです。この引数は、非プリエンティブ・プラットフォームにおいては無視されます。非プリエンティブ・プラットフォームにおいては (CS\_VOID\*)0 に設定してください。

*reserved2*

このパラメータは、現時点では使用されていません。0 に設定してください。

戻り値

**表 3-135 : 戻り値 (srv\_sleep)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local Prototype.
**/
CS_RETCODE      ex_srv_sleep  PROTOTYPE((
CS_VOID         *sleepvnt,
CS_CHAR         *sleeplbl,
CS_INT          *infop
));

/*
** EX_SRV_SLEEP
**
** This routine will suspend the currently executing thread.
**
** Arguments:
**
** sleepvnt A void pointer that srv_wakeup uses to wake up
```

```

**          the thread.
**  sleeplbl  A pointer to a null terminated string that
**            identifies the event being the thread is sleeping
**            on.This is primarily used for debugging.
**  infop     A pointer to a CS_INT that is set to one of the
**            following values:
**            SRV_I_INTERRUPTED - srv_ucwakeup
**            unconditionally woke the thread.
**            SRV_I_UNKNOWN - Some other error occurred.
**
**
** Returns
**
**          CS_SUCCEEDED
**          CS_FAIL
**
*/
CS_RETCODE    ex_srv_sleep(sleepevnt,sleeplbl,infop)
CS_VOID      *sleepevnt;
CS_CHAR      *sleeplbl;
CS_INT       *infop;
{
    /* Check arguments.*/
    if(sleepevnt == (CS_VOID *)NULL)
    {
        return(CS_FAIL);
    }
    /*
    ** Using SRV_M_ATTNWAKE means the thread should wake up
    ** unconditionally if it receives an attention.
    */

    return(srv_sleep(sleepevnt,sleeplbl,SRV_M_ATTNWAKE,infop,(CS_VOID*)0,(CS_V
OID*)0));
}

```

**使用法**

- `srv_sleep` は、現時点で実行しているスレッドを中断し、再スケジューリングを開始します。`srv_wakeup` が同じイベントで呼び出されるまで、スレッドはスリープします。
- `sleepflags` の値によっては、スリープ中のスレッドは、アテンションを受け取ることによって、ウェイクアップすることもできます。
- `srv_sleep` の呼び出しに続く文で、スレッドは実行を再開します。
- `srv_sleep` は、`SRV_START` ハンドラにおいては使用できません。

- `srv_sleep` を、割り込みレベル・コードから呼び出すことはできません。このルールに違反すると、問題が発生する可能性があります。
- 使用しているプラットフォームがプリエンティブ・スケジューリングをサポートしているかどうかを調べるには、`srv_capability` を呼び出してください。
- スレッドがスリープしないうちに起動イベントが発生したら、`reserved1` パラメータはプリエンティブ・スケジューリングで起り得る競合状況を阻止します。プリエンティブ・スケジューリングの例については、使用しているプラットフォーム用の『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

参照

[srv\\_wakeup](#)

## srv\_spawn

説明

サービス・スレッドを割り付けます。

構文

```
CS_RETCODE srv_spawn(sppp, stacksize, funcp,
                    argp, priority)

SRV_PROC    **sppp;
CS_INT      stacksize;
CS_RETCODE  (*funcp)();
CS_VOID     *argp;
CS_INT      priority;
```

パラメータ

*sppp*

スレッド構造体ポインタへのポインタです。呼び出しが成功した場合は、内部のスレッド構造体のアドレスが *sppp* に返されます。

*stacksize*

スタックのサイズです。これは、最低 `SRV_C_MINSTACKSIZE` でなくてはなりません。`srv_props` によって設定されたスタック・サイズ (`SRV_S_STACKSIZE`) を使用するには、`SRV_DEFAULT_STACKSIZE` を指定します。

*funcp*

新しく作成されたスレッドのためのエントリ・ポイントである関数へのポインタです。スレッドは、*funcp* に位置するルーチンを実行することで開始します。このルーチンから戻ったり、`srv_termproc` が呼び出されるとき、スレッドは解放されます。エントリ・ポイント関数ポインタは `SRV_C_START_LISTENER` であることが必要です。

*argp*

スレッドが実行を始めるとき、\**funcp* のルーチンに渡されるポインタです。エントリ関数ポインタ (*funcp*) が

SRV\_C\_START\_LISTENER の場合は、*argp* が CS\_TRANADDR 構造を指す必要があります。

*priority*

生成されたスレッドの基本優先順位を示す SRV\_C\_LOWPRIORITY から SRV\_C\_MAXPRIORITY の間の整数です。デフォルトの優先順位は、SRV\_C\_DEFAULTPRI です。

## 戻り値

スレッドが正常に生成された場合、*srv\_spawn* は CS\_SUCCEED を返します。これは十分な Open Server の内部リソースが使用可能であることだけを保証します。エントリ・ポイント・ルーチンまたはその引数の有効性を検証しません。スレッドが生成できない場合には、*srv\_spawn* は CS\_FAIL を返します。

表 3-136 : 戻り値 (*srv\_spawn*)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <stdio.h>
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE      entryfunc PROTOTYPE((
CS_CHAR        *message
));

CS_RETCODE      ex_srv_spawn PROTOTYPE((
SRV_PROC        *spp,
CS_INT          stacksize,
CS_INT          priority
));

CS_RETCODE      entryfunc(message)
CS_CHAR        *message;
{
    printf("Welcome to a new thread - %s!¥n", message);
    return (CS_SUCCEED);
}
```

```

/*
** EX_SRV_SPAWN
**
**     Example routine to allocate a service thread
**
** Arguments:
**     spp                A pointer to an internal thread control
**     structure.
**     stacks            The desired thread stack size.
**     priority          The desired thread scheduling priority.
**
** Returns:
**
**     CS_SUCCEED
**     CS_FAIL
*/
CS_RETCODE    ex_srv_spawn(spp, stacksize, priority)
SRV_PROC      *spp;
CS_INT        stacksize;
CS_INT        priority;
{
    CS_CHAR     msgarg[20];

    strcpy(msgarg, "come in");

    return(srv_spawn(&spp, stacksize, entryfunc, msgarg,
                    priority));
}

```

使用法

- `srv_spawn` は、イベント駆動型ではなく、どのクライアントとも関連しない「サービス・スレッド」を割り付けます。そのスレッドは、スケジューラの制御の下で実行されます。
- `srv_spawn` によって作成されたスレッドは、共有デバイスやデータ・オブジェクトにアクセスするなどの、イベント駆動型スレッドに要求されるサービスを提供することが多いため、サービス・スレッドと呼ばれます。
- `srv_spawn` は、Open Server に新しいスレッドについて知らせ、そのスレッドを実行可能にします。スレッドは、すぐには実行を開始しません。スレッドが実行を始める時点は、生成スレッドの優先順位や他の実行可能なスレッドの優先順位のような多くの要素によって決定されます。

- `SRV_S_STACKSIZE` で `stacksize` を設定するために `srv_props` を呼び出さない場合は、新しいスレッドがデフォルトの `stacksize` で作成されます。このデフォルトの `stacksize` は、使用するプラットフォームに応じて異なります。Open Server のネイティブスレッド・バージョンでは、基本となるスレッドのデフォルトの `stacksize` が使用されます。
- 複数のスレッドにより実行されたコードは、リエントラントでなくてはなりません。
- `SRV_TLISTENER` スレッド・タイプは動的リスナに使用されます。
- 指定されたホスト名は複数の IPv4 および IPv6 アドレスに変換できます。つまり、動的リスナを起動することで複数のスレッドが作成されることとなります。これらのスレッドの `SRV_PROC` ポインタを取得する唯一の方法は、`SRV_LISTEN_PREBIND` イベントと `SRV_LISTEN_POSTBIND` イベントを使用することです。
- エントリ・ポイント関数ポインタ (*funcp*) が `SRV_C_START_LISTENER` の場合は、*stacksize* と *priority* の両方に `CS_UNUSED` を指定し、*sppp* を `null` に設定する必要があります。

## 参照

[srv\\_callback](#), [srv\\_createproc](#), [srv\\_props](#), [srv\\_termproc](#)

## srv\_symbol

## 説明

Open Server のトークン値を読み込み可能な文字列に変換します。

## 構文

```
CS_CHAR *srv_symbol(type, symbol, lenp)
```

```
CS_INT    type;
CS_INT    symbol;
CS_INT    *lenp;
```

## パラメータ

*type*

トークンのタイプです。表 3-137 に、有効なトークンのタイプを示します。

**表 3-137 : type に対応するトークンのタイプ (srv\_symbol)**

トークンのタイプ	説明
SRV_DATATYPE	データ型
SRV_EVENT	イベント・タイプ
SRV_DONE	DONE ステータス・タイプ
SRV_ERROR	エラー重大度トークン

*symbol*

実際のトークン値です。

*lenp*

返される文字列長を含む CS\_INT 変数へのポインタです。

戻り値

**表 3-138 : 戻り値 (srv\_symbol)**

戻り値	意味
Open Server のトークン値の、読み込み可能な変換である null ターミネータ文字列を指すポインタ	トークン値。
null ポインタ	Open Server が <i>type</i> または <i>symbol</i> を認識しない。 Open Server が <i>lenp</i> を -1 に設定する。

例

```
#include          <ospublic.h>
/*
** Local Prototype
**/
extern CS_RETCODE      ex_srv_symbol PROTOTYPE((
CS_INT type,
CS_INT symbol,
CS_CHAR *namep
));
/*
** EX_SRV_SYMBOL
**
** Retrieve a printable string representation of an Open Server
** symbol
**
** Arguments:
** type          Symbol type
** symbol        Symbol for which to retrieve string
** namep        Return symbol string here
** Returns:
```



```
**      CS_SUCCEEDED      Symbol string was retrieved successfully
**      CS_FAIL           An error was detected
*/
CS_RETCODE      ex_srv_symbol(type, symbol, namep)
CS_INT          type;
CS_INT          symbol;
CS_CHAR         *namep;
{
    CS_INT      len;
    namep = srv_symbol(type, symbol, &len);
    if(namep == (CS_CHAR *)NULL)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEEDED);
}
```

## 使用法

- `srv_symbol` は、Open Server トークン値を記述する、読み込み可能な `null` で終了する文字列へのポインタを返します。
- `srv_symbol` のポインタは、決して上書きされない領域を指すポインタを返すので、同じ文で複数回 `srv_symbol` を呼び出しても安全です。
- [表 3-139](#) は、`srv_symbol` が変換できるトークンをまとめたものです。

表 3-139 : 変換可能なトークン (srv\_symbol)

トークンのタイプ	トークン	説明
SRV_ERROR	SRV_INFO	エラー重大度タイプ
SRV_ERROR	SRV_FATAL_PROCESS	エラー重大度タイプ
SRV_ERROR	SRV_FATAL_SERVER	エラー重大度タイプ
SRV_DONE	SRV_DONE_MORE	DONE パケット・ステータス・フィールド
SRV_DONE	SRV_DONE_ERROR	DONE パケット・ステータス・フィールド
SRV_DONE	SRV_DONE_FINAL	DONE パケット・ステータス・フィールド
SRV_DONE	SRV_DONE_FLUSH	DONE パケット・ステータス・フィールド
SRV_DONE	SRV_DONE_COUNT	DONE パケット・ステータス・フィールド
SRV_DATATYPE	CS_CHAR_TYPE	Char データ型
SRV_DATATYPE	CS_BINARY_TYPE	binary データ型
SRV_DATATYPE	CS_TINYINT_TYPE	1 バイト integer データ型
SRV_DATATYPE	CS_SMALLINT_TYPE	2 バイト integer データ型
SRV_DATATYPE	CS_INT_TYPE	4 バイト integer データ型
SRV_DATATYPE	CS_REAL_TYPE	real データ型
SRV_DATATYPE	CS_FLOAT_TYPE	float データ型
SRV_DATATYPE	CS_BIT_TYPE	bit データ型
SRV_DATATYPE	CS_DATETIME_TYPE	datetime データ型
SRV_DATATYPE	CS_DATETIME4_TYPE	4 バイト datetime データ型
SRV_DATATYPE	CS_MONEY_TYPE	money データ型
SRV_DATATYPE	CS_MONEY4_TYPE	4 バイト money データ型
SRV_DATATYPE	SRVCHAR	Char データ型
SRV_DATATYPE	SRVVARCHAR	可変長 char データ型
SRV_DATATYPE	SRVBINARY	binary データ型
SRV_DATATYPE	SRVVARBINARY	可変長 binary データ型
SRV_DATATYPE	SRVINT1	1 バイト integer データ型
SRV_DATATYPE	SRVINT2	2 バイト integer データ型
SRV_DATATYPE	SRVINT4	4 バイト integer データ型
SRV_DATATYPE	SRVINTN	integer データ型、null 可能
SRV_DATATYPE	SRVBIT	bit データ型
SRV_DATATYPE	SRVDATETIME	datetime データ型

トークンのタイプ	トークン	説明
SRV_DATATYPE	SRVDATETIME4	4 バイト datetime データ型
SRV_DATATYPE	SRVDATETIMN	datetime データ型、null 可能
SRV_DATATYPE	SRVMONEY	money データ型
SRV_DATATYPE	SRVMONEY4	4 バイト money データ型
SRV_DATATYPE	SRVMONEYN	money データ型、null 可能
SRV_DATATYPE	SRVREAL	4 バイト float データ型
SRV_DATATYPE	SRVFLT8	8 バイト float データ型
SRV_DATATYPE	SRVFLTN	8 バイト float データ型、null 可能
SRV_DATATYPE	SRV_LONGCHAR_TYPE	long char データ型
SRV_DATATYPE	SRV_LONGBINARY_TYPE	long binary データ型
SRV_DATATYPE	SRV_TEXT_TYPE	text データ型
SRV_DATATYPE	SRV_IMAGE_TYPE	image データ型
SRV_DATATYPE	SRV_NUMERIC_TYPE	numeric データ型
SRV_DATATYPE	SRV_DECIMAL_TYPE	decimal データ型
SRV_DATATYPE	SRVVOID	void データ型
SRV_EVENT	SRV_ATTENTION	Open Server イベント・タイプ
SRV_EVENT	SRV_BULK	Open Server イベント・タイプ
SRV_EVENT	SRV_CONNECT	Open Server イベント・タイプ
SRV_EVENT	SRV_CURSOR	Open Server イベント・タイプ
SRV_EVENT	SRV_DISCONNECT	Open Server イベント・タイプ
SRV_EVENT	SRV_DYNAMIC	Open Server イベント・タイプ
SRV_EVENT	SRV_LANGUAGE	Open Server イベント・タイプ
SRV_EVENT	SRV_MSG	Open Server イベント・タイプ
SRV_EVENT	SRV_OPTION	Open Server イベント・タイプ
SRV_EVENT	SRV_RPC	Open Server イベント・タイプ

トークンのタイプ	トークン	説明
SRV_EVENT	SRV_START	Open Server イベント・タイプ
SRV_EVENT	SRV_STOP	Open Server イベント・タイプ
SRV_EVENT	SRV_URGDISCONNECT	Open Server イベント・タイプ

参照

[srv\\_descfmt](#)

## srv\_tabcolname

説明 「ブラウズ・モード」の結果カラムに結果テーブルを関連付けます。

構文 CS\_RETCODE srv\_tabcolname(spp, colnum, brwsdescp)

```
SRV_PROC          *spp;
CS_INT            colnum;
CS_BROWSEDESC    *brwsdescp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*colnum*

以前に *srv\_descfmt* を使って記述したカラムを識別するために使用された番号です。

*brwsdescp*

該当するカラムのブラウズ情報を持つ構造体へのポインタです。具体的に、カラムや元のカラム名および名前の長さを含むテーブル（先に *srv\_tabname* で記述された）の番号を含んでいなければなりません。カラムが select 文 (CS\_BROWSEDESC 構造体において CS\_RENAMED のステータスによって指示された) の中で名前変更されていた場合にのみ、元のカラム名と名前の長さが必要になることに注意してください。「CS\_BROWSEDESC 構造体」(55 ページ) を参照してください。

戻り値

表 3-140 : 戻り値 (srv\_tabcolname)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include      <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE    ex_srv_tabcolname PROTOTYPE((
SRV_PROC      *spp,
CS_INT        colnum,
CS_BROWSEDESC *bdp
));

/*
** EX_SRV_TABCOLNAME
**
** Example routine to associate a browse mode result column
** with result tables.
**
** Arguments:
** spp      A pointer to an internal thread control structure.
**
** colnum   The column number.
**
** bdp      A pointer to the browse descriptor for the
** column.
**
** Returns:
** CS_SUCCEED   If we successfully associated this result
**              column with its table.
**
** CS_FAIL      If an error was detected.
**
*/
CS_RETCODE    ex_srv_tabcolname(spp, colnum, bdp)
SRV_PROC      *spp;
CS_INT        colnum;
CS_BROWSEDESC *bdp;
{
    CS_RETCODE    result;
```

```

    result = srv_tabcolname(spp, colnum, bdp);

    return (result);
}

```

使用法

- `srv_tabcolname` は、クライアントにブラウザ・モードの結果情報を送るために使用されます。次にアプリケーションが送ることができる情報を示します。
  - 結果カラムがマップするテーブルの名前。
  - クライアント・クエリの `select` 文内で名前が変更されたカラムの本来の名前。
- カラムはあらかじめ `srv_descfmt` を使って定義されていなければなりません。
- テーブルはあらかじめ `srv_tabname` を使って定義されていなければなりません。
- `srv_tabcolname` は、結果ローのカラムである各結果カラムに対して、一度ずつ呼び出されます。

参照

[srv\\_descfmt](#), [srv\\_tabname](#), 「ブラウザ・モード」 (23 ページ)

## srv\_tabname

説明

一連のブラウザ・モードの結果と関連付けられたテーブルの名前を提供します。

構文

```

CS_RETCODE srv_tabname(spp, tablenum, tablename,
                        namelen)
SRV_PROC    *spp;
CS_INT      tablenum;
CS_CHAR     *tablename;
CS_INT      namelen;

```

パラメータ

*spp*  
内部スレッド制御構造体へのポインタです。

*tablenum*  
以降の `srv_tabcolname` の呼び出しにおいて、テーブルを識別するために使われる番号です。

*tablenamep*

テーブル名へのポインタです。テーブルには必ず名前があるので、名前が null ということはありません。

*namelen*

テーブル名の長さをバイト数で示したものです。*namelen* が CS\_NULLTERM の場合は、Server Library はテーブル名が null で終了するものとみなします。

戻り値

表 3-141 : 戻り値 (srv\_tabname)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include      <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE   ex_srv_tabname PROTOTYPE((
SRV_PROC     *sproc,
CS_INT       tablenum,
CS_CHAR      *tablename
));

/*
** EX_SRV_TABNAME
**   An example routine to provide the name of the table
**   associated with a set of browse mode results.
**
** Arguments:
**   sproc           A pointer to an internal thread control
**   structure.
**
**   tablenum       The number that will be used to identify
**                  the table in subsequent calls to
**
**                  srv_tabcolname.
**   tablename      A null terminated string specifying the
**                  table name.
**
** Returns:
**   CS_SUCCEED     If the table is successfully described.
**   CS_FAIL        If an error was detected.
```

```

*/
CS_RETCODE      ex_srv_tabname(sproc, tablenum, tablename)
SRV_PROC        *sproc;
CS_INT          tablenum;
CS_CHAR         *tablename;
{
    return( srv_tabname(sproc, tablenum, tablename,
                        CS_NULLTERM) );
}

```

**使用法**

- `srv_tabname` は、ブラウザ・モードの結果に対応したテーブルの名前をクライアントに送るのに使用されます。
- Open Server アプリケーションは、ブラウザ・モードの結果に関連する各テーブルに対して、一度ずつ `srv_tabname` を呼び出さなければなりません。
- `tablenum` は、記述されたすべてのテーブルに対してユニークなものでなくてはなりません。テーブルは、いかなる順序でも記述可能です。
- アプリケーションは、ブラウザ・モードの結果カラムを `srv_tabcolname` ルーチンを使って特定の結果のテーブルにリンクします。`srv_tabname` は、必ず `srv_tabcolname` より前に呼び出される必要があります。

**参照**

[srv\\_descfmt](#), [srv\\_tabcolname](#), 「ブラウザ・モード」 (23 ページ)

## srv\_termproc

**説明**

スレッドの実行を中止します。

**構文**

```

CS_RETCODE srv_termproc(spp)
SRV_PROC      *spp;

```

**パラメータ**

*spp*  
内部スレッド制御構造体へのポインタです。



戻り値

表 3-142 : 戻り値 (srv\_termproc)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include          <ospublic.h>
/*
** Local Prototype.
**
*/
CS_RETCODE          ex_srv_termproc PROTOTYPE((
SRV_PROC           *spp
));

/*
** EX_SRV_TERMPROC
**
** Example routine to terminate the execution of a thread using
**   srv_termproc.
**
** Arguments:
**   spp       A pointer to an internal thread control structure.
** Returns:
**
**   CS_SUCCEED   Thread successfully terminated
**   CS_FAIL      An error was detected.
**
*/
CS_RETCODE          ex_srv_termproc(spp)
SRV_PROC           *spp;
{
    /*
    ** Terminate the thread.
    **
    */
    if (srv_termproc(spp) != CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}
```

使用法

- `srv_createproc` を使用すると、Open Server アプリケーションはクライアント接続に関連していないイベント・ハンドラ・ドライバ・スレッドを作成できます。
- `srv_termproc` は、SRV\_START ハンドラでは使用できません。

- `srv_termproc` を割り込みレベル・コードから呼び出さないでください。結果が予想できません。
- ミューテックス、ミューテックス・ロック、レジスタード・プロセスージャ、キューされたイベント、およびスレッドに関連したメッセージは、スレッドが終了すると削除されます。
- 次のコードは、`srv_termproc` の例です。

参照

[srv\\_createproc](#), [srv\\_event](#), [srv\\_event\\_deferred](#), [srv\\_spawn](#)

## srv\_text\_info

説明

`text` または `image` データの記述を設定または取得します。

構文

```
CS_RETCODE srv_text_info(spp, cmd, item, iodescp)
```

```
SRV_PROC      *spp;
CS_INT        cmd;
CS_INT        item;
CS_IODESC     *iodescp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

データ・フローの方向です。表 3-143 に、*cmd* の有効値を示します。

表 3-143 : cmd の値 (srv\_text\_info)

値	意味
CS_SET	Open Server アプリケーションは、text や image データを記述するための内部 Server-Library 構造体を設定している。srv_text_info の呼び出しは、iodescp の情報を使って (Open Server 内部の) text や image データを更新する (アプリケーションは、srv_descfmt を使ってカラムをあらかじめ記述していなければならない)。一般的には、srv_send_text、または srv_bind および srv_xferdata の呼び出しが後に続く。
CS_GET	Open Server はクライアントから読み込む text または image データの全長で iodescp 構造体を更新している。一般的には、srv_get_text の呼び出しが次に続く。CS_GET の方向についての制限については、「使用法」の項を参照。

*item*

記述されるカラムのカラム番号です。ローにある最初のカラムはカラム番号 1 です。cmd が CS\_GET の時、このパラメータは無視されます。

*iodescp*

テキスト・カラムのオブジェクト名、テキスト・ポインタ、およびタイム・スタンプを記述する構造体へのポインタです。詳細については、「CS\_IODESC 構造体」(61 ページ) を参照してください。

## 戻り値

表 3-144 : 戻り値 (srv\_text\_info)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

## 例

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_text_info PROTOTYPE((
SRV_PROC *spp,
CS_INT item
CS_IODESC *iodp
));

/*
** EX_SRV_TEXT_INFO
**
```

```

**      Example routine to set a column's text or image data
**      description before transferring a data row, using
**      srv_text_info. This example routine would be used in a
**      gateway application, where the Open Client application has
**      initiated an update of text or image data.
**
** Arguments:
**      spp          A pointer to an internal thread control structure.
**      item        The column number of the column being described.
**      iodp        A pointer to a CS_IODESC structure that describes the
**                  text or image data (This structure is passed from the
**                  Open Client application).
**
** Returns:
**      CS_SUCCEED   Text or image data successfully described.
**      CS_FAIL      An error occurred was detected.
*/
CS_RETCODE      ex_srv_text_info(spp, item, iodp)
SRV_PROC        *spp;
CS_INT          item;
CS_IODESC       *iodp;
{
    /*
    ** Describe the text or image data for the column.
    */
    if (srv_text_info(spp, (CS_INT)CS_SET, item, iodp) !=
        CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

**使用法**

- `srv_text_info` は、結果ローを送信したりパラメータを取得したりするために、`text` や `image` カラムを記述するために使用されています。
- `cmd` が `CS_GET` の場合、`srv_text_info` は `SRV_BULK` イベント・ハンドラから呼び出さなければなりません。
- `cmd` が `CS_GET` の場合、`srv_text_info` は `srv_get_text` の呼び出しよりも前に、呼び出さなければなりません。
- `cmd` が `CS_SET` の場合、`srv_text_info` は、`srv_xferdata` や `srv_send_text` が呼び出される前に、ローの各 `text` や `image` データ型のカラムに対して呼び出さなければなりません。
- `text` や `image` データは、`srv_bind`、`srv_xferdata`、または `srv_send_text` を使って、クライアントに転送されます。

参照 [srv\\_bind](#), [srv\\_descfmt](#), [srv\\_get\\_text](#), [srv\\_send\\_text](#), [srv\\_xferdata](#), 「[text と image](#)」 (225 ページ)

## srv\_thread\_props

説明 スレッド・プロパティを定義し、取得します。

構文 `CS_RETCODE srv_thread_props(spp, cmd, property, bufp, buflen, outlenp)`

```
SRV_PROC *spp;
CS_INT cmd;
CS_INT property;
CS_VOID *bufp;
CS_INT buflen;
CS_INT *outlenp;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*cmd*

実行するアクションです。表 3-145 に、*cmd* の有効値を示します。

表 3-145 : *cmd* の値 (srv\_thread\_props)

値	意味
CS_SET	Open Server アプリケーションがプロパティを設定している。この場合、 <i>bufp</i> は、プロパティが設定される値を持ち、 <i>buflen</i> は、その値のサイズのバイト数でなければならない。
CS_GET	Open Server アプリケーションがプロパティを取得している。この場合、 <i>bufp</i> は、プロパティ値が置かれているバッファを指し、 <i>buflen</i> は、そのバッファのサイズのバイト数でなければならない。
CS_CLEAR	Open Server アプリケーションは、プロパティをデフォルト値にリセットしている。この場合は、 <i>bufp</i> 、 <i>buflen</i> 、 <i>outlenp</i> は無視される。

*property*

設定、取得、またはクリアされるプロパティです。

*bufp*

クライアントからのプロパティ値の情報が設定されたり、プロパティ値の情報が取得されたりする Open Server アプリケーション・データ・バッファへのポインタです。

*buflen*

バッファ長をバイト数で示したものです。

*outlenp*

取得されたプロパティ値の長さのバイト数を設定するために Open Server が使用する CS\_INT 変数へのポインタです。この引数はオプションであり、*cmd* が CS\_GET のときにのみ使用します。

戻り値

**表 3-146 : 戻り値 (srv\_thread\_props)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <ospublic.h>
/*
** Local Prototype.
**
*/
CS_RETCODE ex_srv_thread_props PROTOTYPE((
SRV_PROC *sp,
CS_CHAR *user,
CS_INT ulen,
CS_INT *lenp
));
/*
** EX_SRV_THREAD_PROPS
**
** Example routine to obtain a client thread's user name through
** srv_thread_props.
**
** Arguments:
** sp A pointer to an internal thread control structure.
** user A pointer to the address of the user name buffer.
** ulen The size of the user name buffer.
** lenp A pointer to an integer variable, that will be set to the length
** of the user name string.
**
** Returns:
** CS_TRUE If the user name was returned succesfully.
** CS_FALSE If an error was detected.
**
*/
CS_RETCODE ex_srv_thread_props(sp, user, ulen, lenp)
SRV_PROC *sp;
CS_CHAR *user;
CS_INT ulen;
```

```

CS_INT      *lenp;
{
    /*
     ** Call srv_thread_props to get the user name.
     */
    if( srv_thread_props(sp, CS_GET, SRV_T_USER, user, ulen, lenp)
        == CS_FAIL )
    {
        /*
         ** An error was al&ready raised.
         */
        return CS_FAIL;
    }
    /*
     ** All done.
     */
    return CS_SUCCEED;
}

```

#### 使用法

- `srv_thread_props` は、スレッド・プロパティを定義、取得、リセットするために呼び出されます。
- 表 3-147 に、有効なプロパティ値、設定や取得の可否、それぞれの値のデータ型を示します。

各スレッド・プロパティの説明については、表 2-28 (169 ページ) を参照してください。

表 3-147 : スレッド・プロパティとそのデータ型 (`srv_thread_props`)

プロパティ	設定/ クリア	取得	cmd が CS_SET のとき の bufp	cmd が CS_GET のときの bufp
SRV_T_APPLNAME	なし	あり	適用しない	文字列へのポインタ
SRV_T_BYTEORDER	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_BULKTYPE	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_BYTEORDER	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_CHARTYPE	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_CLIB	なし	あり	適用しない	文字列へのポインタ
SRV_T_CLIBVERS	なし	あり	適用しない	文字列へのポインタ
SRV_T_CLIENTLOGOUT	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_CONVERTSHORT	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_DUMPLOAD	なし	あり	適用しない	CS_BOOL へのポインタ

プロパティ	設定／ クリア	取得	cmd が CS_SET のとき の bufp	cmd が CS_GET のときの bufp
SRV_T_ENDPOINT	なし	あり	適用しない	終了ポイント (ファイル記述子またはファイル処理) を保持できるサイズのバッファへの CS_VOID ポインタ
SRV_T_EVENT	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_EVENTDATA	なし	あり	適用しない	CS_VOID ポインタのアドレス
SRV_T_FLTTYPE	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_GOTATTENTION	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_HOSTNAME	なし	あり	適用しない	文字列へのポインタ
SRV_T_HOSTPROCID	なし	あり	適用しない	文字列へのポインタ
SRV_T_IODEAD	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_LISTENADDR	なし	あり	適用しない	CS_TRANADDR 構造体へのポインタ
SRV_T_LOCALE	あり	あり	CS_LOCALE ポインタへのポインタ	CS_LOCALE ポインタへのポインタ
SRV_T_LOCALID	あり	なし	適用しない	リスナ SSL 証明書
SRV_T_LOGINTYPE	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_MACHINE	なし	あり	適用しない	文字列へのポインタ
SRV_T_MIGRATED	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_MIGRATE_STATE	なし	あり	適用しない	SRV_MIG_STATE へのポインタ
SRV_T_NEGLOGIN	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_NOTIFYCHARSET	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_NOTIFYDB	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_NOTIFYLANG	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_NOTIFYPND	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_NUMRMTPWDS	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_PACKETSIZE	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_PASSTHRU	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_PRIORITY	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_PWD	なし	あり	適用しない	文字列へのポインタ
SRV_T_REMOTEADDR	なし	あり	適用しない	CS_TRANADDR 構造体へのポインタ



プロパティ	設定／ クリア	取得	cmd が CS_SET のとき の bufp	cmd が CS_GET のときの bufp
SRV_T_RETPARMS	なし	あり	適用しない	実行時にエラーが発生した場合に送信されるリターン・パラメータ
SRV_T_RMTPWDS	なし	あり	適用しない	SRV_RMTPWD 構造体の配列へのポインタ
SRV_T_RMTSERVER	なし	あり	適用しない	文字列へのポインタ
SRV_T_ROWSENT	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_SEC_CHANBIND	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_CONFIDENTIALITY	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_CREDTIMEOUT	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_SEC_DATAORIGIN	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_DELEGATION	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_DELEGCREC	なし	あり	適用しない	CS_VOID へのポインタ
SRV_T_SEC_DETECTREPLAY	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_DETECTSEQ	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_INTEGRITY	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_MECHANISM	なし	あり	適用しない	CS_CHAR へのポインタ
SRV_T_SEC_MUTUALAUTH	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_NETWORKAUTH	なし	あり	適用しない	CS_BOOL へのポインタ
SRV_T_SEC_SESSTIMEOUT	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_SESSIONID	あり	あり	CS_SESSIONID へのポインタ	CS_SESSIONID へのポインタ
SRV_T_SPID	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_STACKLEFT	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_TDSVERSION	あり	あり	CS_INT へのポインタ	CS_INT へのポインタ
SRV_T_TYPE	なし	あり	適用しない	CS_INT へのポインタ
SRV_T_USER	なし	あり	適用しない	文字列へのポインタ
SRV_T_USERDATA	あり	あり	CS_VOID ポインタ	CS_VOID ポインタのアドレス
SRV_T_USERVLANG	あり	あり	CS_BOOL へのポインタ	CS_BOOL へのポインタ

プロパティ	設定／ クリア	取得	cmd が CS_SET のとき の bufp	cmd が CS_GET のときの bufp
SRV_T_USTATE	あり	あり	文字列へのポイン タ	文字列へのポインタ

- [表 3-148](#) に、定義可能なスレッド・プロパティのデフォルト値を示します (CS\_SET)。

**表 3-148: 定義できるスレッド・プロパティとそのデフォルト値 (srv\_thread\_props)**

プロパティ	デフォルト
SRV_T_USERDATA	(CS_VOID *)NULL
SRV_T_USTATE	NULL 文字列
SRV_T_TDSVERSION	最小値 (クライアントおよびサーバのデ フォルト)
SRV_T_USESRVLANG	SRV_S_USESRVLANG の値
SRV_T_LOCALE	(CS_LOCALE *)NULL

- スレッド・プロパティを取得しているとき (CS\_GET)、プロパティ値がユーザ・バッファに入り切らないことを *buflen* が示した場合には、Open Server は、\*outlenp に要求されるバイト数を設定し、アプリケーション・バッファは変更されません。
- 各スレッド・プロパティの説明については、[表 2-28 \(169 ページ\)](#) を参照してください。

参照

[srv\\_props](#), 「プロパティ」 (157 ページ)

## srv\_timedsleep

説明

イベントが通知されるまで、または指定された時間が経過するまでスリープします。srv\_timedsleep が利用できるのはリエントラント・ライブラリのみです。

構文

```
CS_RETCODE srv_timedsleep(sleepevent, sleeplabel,
                           sleepflags, infop, srvmutex, timeout)
```

```
CS_VOID      *sleepevent;
CS_CHAR      *sleeplabel;
CS_INT       sleepflags;
CS_VOID      *infop;
```

	SRV_OBJID	srvmutex;
	CS_INT	timeout;
パラメータ	<i>sleepevent</i>	スリープが発生するイベントへの汎用のポインタです。
	<i>sleeplabel</i>	デバッグを目的とした文字列へのポインタです。
	<i>sleepflags</i>	このパラメータは、現在実行中のスレッドを中断する <code>srv_sleep</code> と同じように使用されます。
	<i>infop</i>	失敗の理由を示す整数へのポインタです。 <i>infop</i> の整数値は次のとおりです。 <ul style="list-style-type: none"> <li>• <code>SRV_I_UNKNOWN</code> – 不明なエラーまたはエラーなし。</li> <li>• <code>SRV_I_TIMEOUT</code> – ルーチンのタイムアウト。</li> <li>• <code>SRV_I_INTERRUPTED</code> – この関数を実行している <code>srvlib</code> プロセスが、<code>srv_ucwakeup()</code> への呼び出しによって中断された。</li> </ul> <hr/> <p><b>注意</b> この関数が <code>SRV_I_INTERRUPTED</code> を返す場合、イベントを待機している間、またはミューテックスをロックしようとしている間に <code>srvlib</code> プロセスが中断されたことを示します。</p> <hr/>
	<i>srvmutex</i>	スリープ時に解放される <code>srvlib</code> ミューテックスです。これは、ウェイクアップ後にロックされます。 <code>srv_timedsleep()</code> がミューテックスの解放とロックを実行しないようにするには、 <code>0</code> を入力してください。
	<i>timeout</i>	ミリ秒単位のタイムアウト時間です。無期限にブロックする場合は <code>0</code> を渡します。

戻り値

**表 3-149 : 戻り値 (srv\_timedsleep)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。詳細については、 <i>info</i> パラメータを参照。

使用法

この関数にミューテックスを渡して、ウェイクアップと同期化させることができます。ミューテックス・ロックを取得した後に `srv_wakeup()` を呼び出す別のスレッドが、このイベントについて、このスリープ関数を実行している `srvlib` プロセスを正常にウェイクアップした時点で、ミューテックスが解放されます。

ルーチンが `CS_SUCCEED` を返すと、`srvlib` ミューテックスはロックされます。`CS_FAIL` を返した場合は、このスレッドによってロックされません。

参照

[srv\\_wakeup](#)

## srv\_ucwakeup

説明

スリープしているスレッドを無条件にウェイクアップします。

構文

```
CS_RETCODE srv_ucwakeup(spp, wakeflags)
SRV_PROC   *spp;
CS_INT     wakeflags;
```

パラメータ

*spp*

内部スレッド制御構造体へのポインタです。

*wakeflags*

`srv_ucwakeup` の動作を変えるビットマスクです。1つのフラグのみ定義されます。使用しない場合は *wakeflags* に 0 を設定します。

`SRV_M_WAKE_INTR`

割り込みレベル・コードからの `srv_ucwakeup` の呼び出しであることを示します。割り込みレベル・コードからの `srv_ucwakeup` 呼び出しを行うときにこのフラグを設定しないと、Open Server アプリケーションは不安定な動作をすることがあります。

戻り値

表 3-150 : 戻り値 (srv\_ucwakeup)

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	スレッドが存在しないか、スリープ中でなかったため、ルーチンが失敗します。

例

```

#include <ospublic.h>
/*
** Local Prototype.
**/

CS_RETCODE      ex_srv_ucwakeup PROTOTYPE((
SRV_PROC        *sproc
));

/*
** EX_SRV_PROC
** An example routine to wake up a sleeping thread from
** a non-interrupt level by using srv_ucwakeup.
**
** Arguments:
** sproc  A pointer to an internal thread control
** structure.
**
** Returns:
** CS_SUCCEED  The specified thread was woken up.
** CS_FAIL    An error was detected.
**/

CS_RETCODE      ex_srv_ucwakeup(sproc)
SRV_PROC        *sproc;
{
/* Wake up the specified thread.*/
return( srv_ucwakeup(sproc, 0));
}

```

使用法

- `srv_ucwakeup` でスレッドをウェイクアップすると、`srv_sleep` は `SRV_I_INTERRUPTED` を返します。
- スレッドを無条件にウェイクアップするには、`srv_ucwakeup` を使用します。デッドロック状態の解除やクリーンアップでは、これが必要な場合があります。
- `srv_ucwakeup` は、`SRV_START` ハンドラにおいては使用できません。

- `srv_ucwakeup` が割込レベル・コードから呼び出された場合は、`wakeflags` を `SRV_M_WAKE_INTR` に設定する必要があります。割込レベル・ルーチンの外部では、`wakeflags` に `SRV_M_WAKE_INTR` を設定しないでください。

参照 [srv\\_sleep](#), [srv\\_wakeup](#), [srv\\_yield](#)

## srv\_unlockmutex

説明 ミューテックスのロックを解除します。

構文 `CS_RETCODE` `srv_unlockmutex(mutexid)`  
`SRV_OBJID` `mutexid`;

パラメータ *mutexid*  
`srv_createmutex` によって返されるユニークなミューテックス識別子です。`mutexid` は、`srv_getobjid` を使ってミューテックス名から得られます。

戻り値 **表 3-151 : 戻り値 (srv\_unlockmutex)**

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

### 例

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_unlockmutex PROTOTYPE((
CS_CHAR *mutex_name
));

/*
** EX_SRV_UNLOCKMUTEX
**
** Example routine to illustrate the use of srv_unlockmutex.
**
** Arguments:
** mutex_name The name of the mutex to be unlocked.
**
```

```

** Returns:
**
**     CS_SUCCEEDED Mutex successfully unlocked.
**     CS_FAIL       An error was detected.
*/
CS_RETURNCODE      ex_srv_unlockmutex(mutex_name)
CS_CHAR            *mutex_name;
{
    SRV_OBJID       id;
    CS_INT          info;

    /* Get the object id for the mutex.*/
    if (srv_getobjid(SRV_C_MUTEX, mutex_name, CS_NULLTERM,
                    &id, &info) == CS_FAIL)
        return(CS_FAIL);

    /* Call srv_unlockmutex to unlock it.*/
    if (srv_unlockmutex(id) == CS_FAIL)
        return(CS_FAIL);

    return (CS_SUCCEEDED);
}

```

**使用法**

- ミューテックス (相互排除セマフォ) をロック解除するとセマフォのロックが解除されて、他のスレッドがミューテックスにアクセスできるようになります。
- `srv_unlockmutex` は、`SRV_START` ハンドラでは使えません。

**参照**

[srv\\_createmutex](#), [srv\\_deletemutex](#), [srv\\_getobjid](#)

## srv\_version

**説明**

アプリケーションが使用している Open Server のバージョンを定義します。

**構文**

```

CS_RETURNCODE srv_version(contextp, version)
CS_CONTEXT    *contextp;
CS_INT        version;

```

パラメータ

*contextp*

アプリケーションが `cs_ctx_alloc` を呼び出して取得した `CS_CONTEXT` 構造体へのポインタです。 `CS_CONTEXT` 構造体は、クライアント・ライブラリと共有した、サーバワイドな設定構造体の役割を果たしています。「[CS-Library](#)」(62 ページ) を参照してください。

*version*

アプリケーションが有効であると仮定する Open Server のバージョンです。

戻り値

**表 3-152 : 戻り値 (srv\_version)**

戻り値	意味
CS_SUCCEEDED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

例

```
#include <stdio.h>
#include <ospublic.h>
.....
/*
 ** This code fragment sets the Open Server version.
 */
main()
{
CS_CONTEXT *cp;
if(cs_ctx_alloc(CS_VERSION_155, &cp) != CS_SUCCEEDED)
{
    fprintf(stderr, "cs_ctx_alloc failed %Yn");
    exit(1);
}
if(srv_version(cp, CS_VERSION_155) != CS_SUCCEEDED)
{
    /*
     ** Release the context structure already allocated.
     */
    (CS_VOID)cs_ctx_drop(cp);
    (CS_VOID)fprintf(stderr, "srv_version failed %Yn");
    exit(1);
}
.....
}
```



使用法	<ul style="list-style-type: none"> <li>• Open Server アプリケーションは、他の Server-Library ルーチンを呼び出す前に、<code>srv_version</code> を呼び出さなければなりません。また、<code>srv_version</code> の呼び出しは CS-Library ルーチンの <code>cs_ctx_alloc</code> を呼び出してから行う必要があります。</li> <li>• アプリケーションは、最初に、<code>cs_config</code> を使用して <code>CS_CONTEXT</code> 構造体にローカライゼーション構成パラメータを設定できます。</li> </ul>
参照	<code>cs_ctx_alloc</code> 、 <code>cs_ctx_props</code>

## srv\_wakeup

説明	スリープ中のスレッドの実行を可能にします。
構文	<pre>CS_RETCODE srv_wakeup(sleepeventp, wakeflags,                       reserved1, reserved2)  CS_VOID     *sleepeventp; CS_INT      wakeflags; CS_VOID     *reserved1; CS_VOID     *reserved2;</pre>
パラメータ	<p><i>sleepeventp</i> スレッドがスリープ中のオペレーティング・システム・イベントの汎用の void ポインタです。</p> <p><i>wakeflags</i> <code>srv_wakeup</code> の動作を変えるビットマスクです。ビットが設定されていない場合、デフォルトの動作は、イベント上でスリープしているすべてのスレッドをウェイクアップさせることです。ビットは論理和をとることができます。表 3-153 に、<i>wakeflags</i> の有効値を示します。</p>

**表 3-153 : wakeflags の値 (srv\_wakeup)**

値	説明
SRV_M_WAKE_INTR	srv_wakeup の呼び出しは、割り込みレベル・コードから行う。割り込みレベル・コードからの srv_wakeup 呼び出しを行うときにこのフラグを使用しないと、Open Server アプリケーションは不安定な動作をすることがある。非割り込みレベルでのこのフラグを使用すると、Open Server アプリケーションは不安定な動作をする。
SRV_M_WAKE_FIRST	イベント上でスリープしている最初のスレッドだけが実行可能にされる。
SRV_M_WAKE_ALL	イベント上でスリープしているすべてのスレッドをウェイクアップさせる。

*reserved1*

このパラメータは、使用されません。(CS\_VOID\*)0 に設定してください。

*reserved2*

このパラメータは、使用されません。(CS\_VOID\*)0 に設定してください。

戻り値

srv\_wakeup は、イベントのスリープ中のスレッドが見つからなかった場合やパラメータがエラーの場合は、CS\_FAIL を返します。1 つまたは複数のスリープ中のスレッドが見つかった場合は、srv\_wakeup は CS\_SUCCEED を返します。

**表 3-154 : 戻り値 (srv\_wakeup)**

戻り値	意味
CS_SUCCEED	1 つまたは複数のスリープ中のスレッドが見つかり、実行可能だった。
CS_FAIL	ルーチンが失敗。またはスリープ中のスレッドが見つからなかった。

例

```
#include <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_wakeup PROTOTYPE((
CS_VOID *sep
));
```

```

/*
** EX_SRV_WAKEUP
**
** Example routine using srv_wakeup to make all Open Server
** threads, which were previously sleeping on the specified
** sleep event, runnable again.
**
** Arguments:
** sep    A generic void pointer, which was used previously in
**        calls to srv_sleep to suspend threads.
**
** Returns:
** CS_SUCCEED    Threads sleeping on the specified sleep event
**               are runnable again.
** CS_FAIL       An error was detected.
*/
CS_RETCODE      ex_srv_wakeup(sep)
CS_VOID        *sep;
{
    /*
    ** Wake up threads for the specified sleep event, passing
    ** zero for reserved fields.
    */
    if (srv_wakeup(sep, (CS_INT)SRV_M_WAKE_ALL,
                   (CS_VOID*)0, (CS_VOID*)0) != CS_SUCCEED)
    {
        return(CS_FAIL);
    }
    return (CS_SUCCEED);
}

```

#### 使用法

- `srv_wakeup` は、*sleepevent* でスリープ中のスレッドをウェイクアップさせます。
- `srv_wakeup` が割り込みレベル・コードから呼び出される場合、実際のウェイクアップは、スケジューラが次に実行するまで遅延されます。
- `srv_wakeup` は、`SRV_START` ハンドラにおいては使用できません。
- Open Server でプリエンティブ・プログラムを書く場合、`srv_wakeup` や `srv_sleep` は、プラットフォームに依存するミューテックスを使わなければなりません。プリエンティブ・スケジューリングの例については、使用しているプラットフォーム用の『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

参照 [srv\\_sleep](#)

## srv\_xferdata

**説明** クライアントにパラメータやデータを送信するか、またはクライアントからパラメータやデータを受信します。

**構文** CS\_RETCODE srv\_xferdata(spp, cmd, type)

```
SRV_PROC      *spp;
CS_INT        cmd;
CS_INT        type;
```

**パラメータ** *spp*  
内部スレッド制御構造体へのポインタです。

*cmd*  
データがクライアントへ送信されているのか、それともクライアントから入ってきているのかを示します。表 3-155 に、*cmd* の有効値を示します。

**表 3-155 : cmd の値 (srv\_xferdata)**

値	説明
CS_SET	アプリケーションは、クライアントにデータを送信するために <i>srv_xferdata</i> を呼び出している。
CS_GET	アプリケーションは、クライアントからのデータを取得するために <i>srv_xferdata</i> を呼び出している。

*type*  
プログラム変数に保管されたり、プログラム変数から読み取られるデータの型です。表 3-156 に、有効な型とその適切なコンテキストを示します。

表 3-156 : type の値 (srv\_xferdata)

Type	有効な cmd	データの内容
SRV_RPCDATA	CS_SET または CS_GET	RPC パラメータ
SRV_ROWDATA	CS_SET のみ	結果ロー・カラム
SRV_CURDATA	CS_GET のみ	カーソル・パラメータ
SRV_KEYDATA	CS_GET のみ	カーソル・キー・カラム
SRV_ERRORDATA	CS_SET のみ	エラー・メッセージ・パラメータ
SRV_DYNDATA	CS_SET または CS_GET	動的 SQL パラメータ
SRV_NEGDATA	CS_SET または CS_GET	ネゴシエーション・ログイン・パラメータ
SRV_MSGDATA	CS_SET または CS_GET	メッセージ・パラメータ
SRV_LANGDATA	CS_GET のみ	言語パラメータ

戻り値

表 3-157 : 戻り値 (srv\_xferdata)

戻り値	意味
<i>CS_SUCCEED</i>	ルーチンが正常に終了した。
<i>CS_FAIL</i>	ルーチンが失敗。
<i>CS_PASSTHRU_MORE</i>	(バッチされたパラメータの場合)パラメータがコピーされました。バッチには、さらにパラメータが存在します。
<i>CS_PARAMS_END</i>	(バッチされたパラメータの場合)パラメータがコピーされました。これがバッチ内の最後のパラメータ・セットです。

例

```
#include          <ospublic.h>
/*
** Local Prototype.
*/
CS_RETCODE      ex_srv_xferdata PROTOTYPE((
SRV_PROC        *spp
));

/*
** EX_SRV_XFERDATA
**
** This routine will send error message parameters to the
** specified client.
**
** Arguments:
```

```

**
**      spp      A pointer to an internal thread control structure.
**
** Returns
**
**      CS_SUCCEED
**      CS_FAIL
**
*/
CS_RETCODE      ex_srv_xferdata(spp)
SRV_PROC        *spp;
{
    /* Check arguments.*/
    if (spp == (SRV_PROC *)NULL)
    {
        return(CS_FAIL);
    }
    return(srv_xferdata(spp,CS_SET,SRV_ERRORDATA));
}

```

#### 使用法

- **srv\_xferdata** は、クライアントにパラメータまたはローのデータを送信したり (**CS\_SET**)、クライアントからパラメータまたはキー・データを取得したりするために使用します。具体的には、データをローカル・プログラム変数から取り出してネットワークを通してクライアントに送信したり (**CS\_SET**)、逆にクライアントからネットワークを通してデータをローカル・プログラム変数に設定したりします (**CS\_GET**)。
- クライアントに表示しなければならない形のデータ (**CS\_SET**) やクライアントに表示した形のデータ (**CS\_GET**) は、事前に **srv\_descfmt** を使って記述しておく必要があります。また、アプリケーションは、ローカル・プログラム変数を定義するために **srv\_bind** を事前に呼び出しておく必要もあります。
- **srv\_xferdata** は、各パラメータ・ストリーム (**CS\_GET**、**CS\_SET**) に対して一度ずつ、または各データ・ロー (**CS\_SET**) に対して一度ずつ呼び出される必要があります。
- バッチされたパラメータのサポートを有効にするには、**SRV\_S\_PARAM\_BATCHING** サーバ・プロパティを **CS\_TRUE** に設定します。それによって、**srv\_xferdata()** は複数のパラメータ・セットを受信できます。たとえば、**srv\_run()** の前に次のように指定します。

```

if (srv_props(ctos_ctx->cx_context, CS_SET,
SRV_S_PARAM_BATCHING, (CS_VOID *)&cs_true,
sizeof(cs_true), NULL) != CS_SUCCEED)

```

```
{...}
```

参照 [srv\\_bind](#), [srv\\_descfmt](#)

## srv\_yield

説明 他のスレッドの実行を可能にします。

構文 `CS_RETCODE srv_yield()`

戻り値 なし。

例

```
#include <stdio.h>
#include          <ospublic.h>

/*
** Local Prototype.
*/
CS_RETCODE ex_srv_yield PROTOTYPE((
    ));
/*
** EX_SRV_YIELD
**
** Example routine to suspend the current thread.
** Arguments:
**     None.
**
** Returns:
**
**     CS_SUCCEEDED
**     CS_FAIL
*/
CS_RETCODE    ex_srv_yield()
{
    printf("I'll wait this one out...¥n");
    if (srv_yield() == CS_FAIL)
    {
        printf("srv_yield() failed.¥n");
        return(CS_FAIL);
    }
    else
    {
        printf("I'm back!¥n");
    }
}
```

```

        return (CS_SUCCEED);
    }
}

```

使用法

- `srv_yield` は、現在のスレッドを中断し、同じかそれ以上の優先順位を持った他の実行可能なスレッドを起動させます。そのスレッドは、後で再スケジュールされます。
- `srv_yield` は、基本的には非プリエンプティブ・スケジューリング時に役立ちます。
- スレッドが `srv_yield` を呼び出して、確立されている新しいスレッドを起動する場合、次のように動作します。
  - a `Open Server` は新しいスレッドの確立を完了します。
  - b 新しいスレッドが実行可能にならない場合、その新しいスレッドは制御されなくて、即座に現在のスレッドが再び制御されるようになると考えられます。

「マルチスレッド・プログラミング」(123 ページ)を参照してください。

- `srv_yield` を呼び出すスレッドが、`srv_yield` の後に続く文を受けて実行を再開します。
- `srv_yield` は、`SRV_START` ハンドラでは使用できません。
- 割り込みレベル・コードから `srv_yield` は呼び出さないでください。

参照

[srv\\_sleep](#), [srv\\_wakeup](#)



# システム・レジスタード・プロシージャ

この章では、Server-Library の各システム・レジスタード・プロシージャについて説明します。システム・レジスタード・プロシージャは、Open Server に組み込まれているレジスタード・プロシージャです。サーバは、これらのプロシージャをすべての Open Server ランタイム・システムで使えるように、初期化のときに登録します。各プロシージャのページでは、パラメータ、それぞれが返す結果、メッセージについて説明します。

システム・レジスタード・プロシージャの詳細については、「[レジスタード・プロシージャ](#)」(185 ページ)を参照してください。

システム・レジスタード・プロシージャ	ページ
<a href="#">sp_ps</a>	505
<a href="#">sp_regcreate</a>	508
<a href="#">sp_regdrop</a>	515
<a href="#">sp_reglis</a>	516
<a href="#">sp_regnowatch</a>	517
<a href="#">sp_regwatch</a>	518
<a href="#">sp_regwatchlist</a>	520
<a href="#">sp_serverinfo</a>	520
<a href="#">sp_terminate</a>	521
<a href="#">sp_who</a>	523

## sp\_ps

### 説明

指定された Open Server のスレッドの詳細なステータス情報を返します。

### 構文

```
sp_ps [loginame | ' spid' ]
```

### パラメータ

*loginame*

ユーザのログイン名です。

*spid*

レポートの対象となるスレッドの内部 ID 番号です。*spid* は、以前の *sp\_who* または *sp\_ps* 呼び出しから得られます。デフォルトでは、すべてのスレッドがリストされます。

例

```
1>execute utility...sp_ps
2>go
```

spid	Login Name	Host Name	Program Name	Task Type	...	
1				SERVER TASK	...	
2				SERVER TASK	...	
3				SERVER TASK	...	
4				SERVICE TASK	...	
11		hiram		SITE HANDLER TASK	...	
14	bud	sonoma	isql	CHILD TASK	...	
...	Status	Sleep Event	Sleep Label	Current Command	...	
...	runnable	369448		NETWORK HANDLER	...	
...	sleeping	369544	MSG AVAILABLE	CONNECT HANDLER	...	
...	sleeping	369640	MSG AVAILABLE	DEFERRED HANDLER	...	
...	runnable	0		SCHEDULER	...	
...	sleeping	369736	MSG AVAILABLE		...	
...	running	416480			...	
...	Blocked	Run	Current	Stack	Net	Net
...	By	Ticks	Priority	Origin	Writes	Reads
...	0	0	8	2794336	0	0
...	0	0	8	2810792	0	0
...	0	0	8	2827184	0	0
...	0	0	15	2843576	0	0
...	0	0	8	2859968	2	7
...	0	0	8	2909208	3	0

この例は、*sp\_ps* プロシージャからの *isql* 出力を示しています。なおこのレポートは出力の都合上、途中を省略しています。

## 使用法

- *sp\_ps* は、指定したサーバ・スレッドまたはすべての現在の Open Server スレッドのステータスを詳細にレポートします。この情報は、アプリケーション開発中のデバッグ作業に役立ちます。
- *loginame* および *spid* は文字列パラメータです。Adaptive Server Enterprise からのリモート・プロシージャ・コールとして *isql* を使用し、*sp\_ps* を実行するときは、構文エラーを避けるために、*spid* を一重引用符 (') で囲みます。

- *loginame* または *spid* を指定しない場合、*sp\_ps* はすべての現在のスレッドをリストします。
- 表 4-1 は、*sp\_ps* が返す情報を示します。

表 4-1 : 返される情報 (sp\_ps)

情報の種類	意味
spid	スレッド内部のスレッド番号。
Login Name	ログインしたユーザの名前。クライアント・スレッドにのみ適用される。
Host Name	クライアント・タスクでは、これがクライアント・マシン名となる。サイト・ハンドラとサーバ間の RPC 接続では、これがリモート Adaptive Server Enterprise 名となる。
Program Name	クライアント・アプリケーション・プログラムの名前。
Task Type	スレッドのタイプ。カラムの有効値は、NETWORK、CLIENT、SERVER、SITE HANDLER、CHILD、SERVICE、UNKNOWN。
Status	スレッドの現在のステータス。カラムの有効値は、running、runnable、sleeping、sick、free、stopped、spawned、terminal、unknown。1 つしかない “running” タスクは、 <i>sp_ps</i> を実行しているスレッド。
Sleep Event	スリープ・スレッドを実行可能にするイベント。
Sleep Label	スリープ・イベントを記述する文字列ラベル。
Current Command	スレッドのステータスを記述する文字列。カラムの内容は <i>srv_thread_props</i> ルーチンで設定される。
Blocked By	(現在は使用されていない)
Run Ticks	(現在は使用されていない)
Current Priority	スレッド実行の優先順位。
Stack Origin	スレッドのスタックが始まるメモリのアドレス。
Net Writes	スレッド開始以後のネットワークの書き込み数。この数字は、サイト・ハンドラとクライアント・スレッドにだけ適用する。
Net Reads	スレッド開始以後のネットワークの読み込み数。この数字は、サイト・ハンドラとクライアント・スレッドにだけ適用する。

表 4-2 は、各カラムにローの形式で返される結果を示しています。

表 4-2 : 返される情報のフォーマット (sp\_ps)

カラム名	データ型	長さ
spid	CS_INT_TYPE	4
Login Name	CS_CHAR_TYPE	SRV_MAXNAME
Host Name	CS_CHAR_TYPE	SRV_MAXNAME
Program Name	CS_CHAR_TYPE	SRV_MAXNAME
Task Type	CS_CHAR_TYPE	SRV_MAXNAME
Status	CS_CHAR_TYPE	SRV_MAXNAME
Sleep Event	CS_INT_TYPE	4
Sleep Label	CS_CHAR_TYPE	SRV_MAXNAME
Current Command	CS_CHAR_TYPE	SRV_MAXNAME
Blocked By	CS_INT_TYPE	4
Run Ticks	CS_INT_TYPE	4
Current Priority	CS_INT_TYPE	4
Stack Origin	CS_INT_TYPE	4
Net Writes	CS_INT_TYPE	4
Net Reads	CS_INT_TYPE	4

参照

[sp\\_terminate](#), [sp\\_who](#)

## sp\_regcreate

説明

Open Server 上にレジスタード・プロシージャを作成するため。

構文

sp\_regcreate proc\_name, parm1, parm2, ...

パラメータ

*proc\_name*

作成するレジスタード・プロシージャ名を指定します。

*parm1, parm2, ...*

(オプション) クライアント・アプリケーションが追加パラメータを渡す場合、ここには新しいプロシージャのパラメータの名前、データ型、デフォルト値を指定します。

例

Client-Library クライアントから sp\_regcreate を呼び出す  
この例では、次のパラメータを必要とするレジスタード・プロシージャ np\_test を作成します。

- 最初のパラメータは *@p1* であり、CS\_INT データ型です。デフォルト値はありません(つまり、この値はデフォルトで NULL に設定されます)。
- 2番目のパラメータは *@p2* であり、CS\_CHAR データ型です。デフォルト値は “No value given” です。
- 3番目のパラメータは *@p3* であり、CS\_INT データ型です。デフォルト値は 0 (ゼロ) です。

この例では、プロシージャを作成する関数 `np_create` と RPC コマンドの結果を処理する関数 `rpc_results` のコードが指定されています。この例では (`rpc_results` を使用して呼び出される) 関数 `ex_fetch_data` は示されていません。この関数は、Client-Library のサンプル・プログラム内のファイル `exutils.c` で定義されています。

```

/*
** np_create() -- Example function to create a notification
** procedure on an Open Server.
**
** Parameters:
** cmd - Command handle for sending commands.
**
** Returns:
** CS_SUCCEED - The notification procedure was successfully
** created.
** CS_FAIL - Couldn't do it.This routine fails if the
** registered procedure already exists.
*/
CS_RETCODE np_create(cmd)
CS_COMMAND *cmd;
{
    CS_DATAFMT    datafmt;
    CS_INT        intval;
    CS_CHAR       charbuf[512];
    CS_BOOL       ok = CS_TRUE;
/*
** Build up an RPC command to create the notification
** procedure np_test, defined as follows:
** np_test @p1 = <integer value>,
**          @p2 = <character value>,
**          @p3 = <integer value>
*/
    if (ok
        && (ct_command(cmd, CS_RPC_CMD,
                      "sp_regcreate", CS_NULLTERM,
                      CS_UNUSED) != CS_SUCCEED))

```

```
    ok = CS_FALSE;
/*
** Name of the created procedure will be 'np_test'.
*/
strcpy(datafmt.name, "proc_name");
datafmt.namelen = strlen(datafmt.name);
datafmt.datatype = CS_CHAR_TYPE;
datafmt.status = CS_INPUTVALUE;
datafmt.maxlength = 255;
strcpy(charbuf, "np_test");
if (ok &&
    ct_param(cmd, &datafmt,
             (CS_VOID *)charbuf, strlen(charbuf), 0)
    != CS_SUCCEED)
{
    fprintf(stdout, "np_create:ct_param() @proc_name failed¥n");
    ok = CS_FALSE;
}
/*
** First parameter is named '@p1', is integer type, and has
** no default (i.e., defaults to NULL).We pass -1 as the
** indicator to ct_param() to specify a NULL value.
*/
strcpy(datafmt.name, "@p1");
datafmt.namelen = strlen(datafmt.name);
datafmt.datatype = CS_INT_TYPE;
datafmt.status = CS_INPUTVALUE;
datafmt.maxlength = CS_UNUSED;
if (ok &&
    ct_param(cmd, &datafmt, (CS_VOID *)NULL, CS_UNUSED, -1)
    != CS_SUCCEED)
{
    fprintf(stdout, "np_create:ct_param() @p1 failed¥n");
    ok = CS_FALSE;
}
/*
** Second parameter is named '@p2', is character type, and has
** default "No value given" .
*/
strcpy(datafmt.name, "@p2");
datafmt.namelen = strlen(datafmt.name);
datafmt.datatype = CS_CHAR_TYPE;
datafmt.status = CS_INPUTVALUE;
datafmt.maxlength = 255;
strcpy(charbuf, "No value given");
if (ok &&
```

```

        ct_param(cmd, &datafmt,
                (CS_VOID *)&charbuf, strlen(charbuf), 0)
        != CS_SUCCEEDED)
    {
        fprintf(stdout, "np_create:ct_param() @p2 failed¥n");
        ok = CS_FALSE;
    }
}
/*
** Third parameter is named '@p3', is integer type, and
** has default 0 (zero).
*/
strcpy(datafmt.name, "@p3");
datafmt.namelen = strlen(datafmt.name);
datafmt.datatype = CS_INT_TYPE;
datafmt.status = CS_INPUTVALUE;
datafmt.maxlength = CS_UNUSED;
intval = 0;
if (ok &&
    ct_param(cmd, &datafmt, (CS_VOID *)&intval, CS_UNUSED, 0)
    != CS_SUCCEEDED)
    {
        fprintf(stdout, "np_create:ct_param() @p3 failed¥n");
        ok = CS_FALSE;
    }
}
/*
** Send the RPC command.
*/
if (ok && ct_send(cmd) != CS_SUCCEEDED)
    ok = CS_FALSE;

/*
** Process the results from the RPC execution.
*/
if (ok && rpc_results(cmd, CS_FALSE) != CS_SUCCEEDED)
    ok = CS_FALSE;

return (ok ?CS_SUCCEEDED :CS_FAIL);

} /* np_create */

/*
** rpc_results() -- Process results from an rpc.
**
** Parameters
**   cmd -- The command handle with results pending.
**   expect_fetchable -- CS_TRUE means fetchable results
**                      are expected.They will be printed w/ the

```

```
**          ex_fetch_data() routine (defined in file exutils.c).
**          CS_FALSE means fetchable results cause this routine
**          to fail.
**
** Returns
**  CS_SUCCEED -- no errors.

**  CS_FAIL -- ct_results failed, returned a result_type value
**           of CS_CMD_FAIL, or returned unexpected fetchable results.
*/
CS_RETCODE rpc_results(cmd, expect_fetchable)
CS_COMMAND      *cmd;
CS_BOOL        expect_fetchable;
{
    CS_RETCODE    results_ret;
    CS_INT        result_type;
    CS_BOOL       ok = CS_TRUE;
    CS_BOOL       cmd_failed = CS_FALSE;
    while (ok &&
           (results_ret
            = ct_results(cmd, &result_type))
           == CS_SUCCEED)
    {
        switch ((int)result_type)
        {
            case CS_STATUS_RESULT:
            case CS_ROW_RESULT:
            case CS_COMPUTE_RESULT:
            case CS_PARAM_RESULT:
                /*
                 ** These cases indicate fetchable results.
                 */
                if (expect_fetchable)
                {
                    /* ex_fetch_data() is defined in exutils.c */
                    ok = (ex_fetch_data(cmd) == CS_SUCCEED);
                }
                else
                {
                    (CS_VOID)fprintf(stdout,
                                     "RPC returned unexpected result%Yn");
                    (CS_VOID)ct_cancel(NULL, cmd, CS_CANCEL_ALL);
                    ok = CS_FALSE;
                }
                break;
            case CS_CMD_SUCCEED:

```



```

    case CS_CMD_DONE:
        /* No action required */
        break;

    case CS_CMD_FAIL:
        (CS_VOID)fprintf(stdout,
            "RPC command failed on server.¥n");
        cmd_failed = CS_TRUE;
        break;

    default:
        /*
         ** Unexpected result type.
         */
        (CS_VOID)fprintf(stdout,
            "RPC returned unexpected result¥n");
        (CS_VOID)ct_cancel(NULL, cmd, CS_CANCEL_ALL);
        ok = CS_FALSE;
        break;
} /* switch */
} /* while */

switch( (int) results_ret)
{
    case CS_END_RESULTS:
    case CS_CANCELED:
        break;
    case CS_FAIL:
    default:
        ok = 0;
}
return ((ok && !cmd_failed) ?CS_SUCCEED :CS_FAIL);

} /* rpc_results() */

```

DB-Library クライアントから `sp_regcreate` を呼び出す

この例では、パラメータを2つ必要とする `pricechange` という名前のレジスタード・プロシージャを作成します。最初のパラメータは `@current_price` で、SYBMONEY データ型で表現されています。2つ目のパラメータは `@sequence_num` で、SYBINT4 データ型です。どちらのパラメータにもデフォルト値はありません。

```

dbnpdefine(dbproc, "pricechange", DBNULLTERM);
dbregparam(dbproc, "@current_price", DBNULLTERM,
    SYBMONEY, DBNODEFAULT, NULL);
dbregparam(dbproc, "@sequence_num", DBNULLTERM,
    SYBINT4, DBNODEFAULT, NULL);
status = dbnpcreate (dbproc);

```

```

if (status == FAIL)
{
    fprintf(stderr,
        "Could not create pricechange procedure.¥n");
}

```

表 4-3 は、SRV\_C\_PROCEXEC コールバック・ハンドラが、pricechange プロシージャが登録中であることを調べるのに使用する呼び出しについてまとめたものです。

表 4-3 : 戻り値 (sp\_regcreate)

関数呼び出し	戻り値
srv_procname(srvproc, (int *) NULL)	“sp_regcreate”
srv_rpcparams(srvproc)	3
srv_paramdata(srvproc, 1)	“pricechange”
srv_paramdata(srvproc, 2)	“@current_price”
srv_paramdata(srvproc, 3)	“@sequence_num”

## 使用法

- クライアント・アプリケーションはリモートで sp\_regcreate を呼び出して、レジスタード・プロシージャを作成します。
- クライアント・アプリケーションが作成するレジスタード・プロシージャは、「ノーティフィケーション (通知) プロシージャ」と呼ばれます。これらのプロシージャにはアプリケーションが定義するコードを指定できません。また、これらのプロシージャは、レジスタード・プロシージャ・ノーティフィケーションに依存するクライアント・アプリケーションに有効です。
- sp\_regcreate の最初のパラメータ (*proc\_name*) は、作成するプロシージャの名前です。新しいレジスタード・プロシージャがパラメータを必要とする場合、そのパラメータは追加パラメータを渡すことによって定義されます。新しいプロシージャの最初のパラメータは sp\_regcreate の 2 番目のパラメータとして、2 番目のパラメータは sp\_regcreate の 3 番目のパラメータとして、というように渡されます。
- Client-Library を使用して構築されたクライアント・アプリケーションは、sp\_regcreate を呼び出す RPC コマンドを送信して、レジスタード・プロシージャを作成できます。

この例は、「Client-Library クライアントから sp\_regcreate を呼び出す」に示してあります。

- DB-Library プログラムは `dbnpdefine`、`dbregparam`、および `dbnpcreate` を使用して、レジスタード・プロシージャを作成します。`dbnpdefine` は内部的に RPC コマンドを生成して、リモートで `sp_regcreate` を呼び出します。`dbnpcreate` は RPC を送信して、その結果を処理します。

この例は、「DB-Library クライアントから `sp_regcreate` を呼び出す」に示してあります。

- Server-Library プログラムは `srv_regdefine`、`srv_regparam`、および `srv_regcreate` を使用して、レジスタード・プロシージャを作成できます。

#### メッセージ

`sp_regcreate` は、次のメッセージを返すことができます。

番号	重大度	テキスト
16505	0	プロシージャが正しく登録された。
16506	11	プロシージャはすでに登録されている。
16507	11	プロシージャを登録できない。

#### 参照

[sp\\_regdrop](#), [sp\\_regnowatch](#), [sp\\_regwatch](#), [srv\\_regdefine](#), [srv\\_regexec](#), [srv\\_reginit](#), [srv\\_regparam](#)

## sp\_regdrop

#### 説明

レジスタード・プロシージャ・リストからプロシージャを削除します。

#### 構文

```
sp_regdrop proc_name
```

#### パラメータ

*proc\_name*

削除するレジスタード・プロシージャの名前です。

#### 例

```
1>execute stock...sp_regdrop pricechange
2>go
```

この例では、`isql` で Adaptive Server Enterprise にログインしたクライアントは、サーバ間のリモート・プロシージャ・コールを使って Open Server stock で `sp_regdrop` を実行します。このプロシージャは、stock から `pricechange` レジスタード・プロシージャを削除します。

```
dbrpcinit(dbproc, "sp_regdrop", NULL);
dbrpcparam(dbproc, "proc_name", NULL, SYBCHAR, -1,
11, "pricechange");
dbrpcsend(dbproc);
```

この例では、DB-Library RPC のルーチンを使用し、単一のパラメータ “pricechange” に対して `sp_regdrop` を実行しています。これにより、`sp_regdrop` システム・プロシージャは、Open Server から pricechange レジスタード・プロシージャを削除します。

#### 使用法

- プロシージャの登録が解除されると、未処理の通知 (ノータフィケーション) 要求があるクライアントは、そのプロシージャはもはや登録されていないというメッセージを受け取ります。
- `sp_regdrop` は、クライアントが `dbnpdrop` を実行した時に実行されます。SRV\_C\_PROCEXEC コールバック・ハンドラは、`sp_regdrop` が実行中であるかどうかを調べるのに `srv_rpcname` を使用できません。次に `srv_bind` および `srv_xferdata` を使用し、最初のパラメータである `proc_name` のポインタを入手できます。

#### メッセージ

```
proc_name has been unregistered.
```

`proc_name` パラメータで指定されたプロシージャは登録を解除されました。

```
proc_name is not a registered procedure.
```

`proc_name` パラメータで指定されたプロシージャは、Open Server で登録されたものではありません。

```
Unable to unregister proc_name.
```

Open Server は、何らかの理由でプロシージャを登録解除することができませんでした。

#### 参照

[sp\\_regdrop](#), [srv\\_regexec](#), [srv\\_reginit](#), [srv\\_regparam](#)

## sp\_reglst

#### 説明

Open Server 上のすべてのレジスタード・プロシージャをリストします。

#### 構文

```
sp_reglst
```

#### 例

```
1>execute utility...sp_reglst
2>go

Procedure Name
-----
sp_who
```

```

sp_regwatch
sp_ps
sp_regdrop
sp_reglst
sp_regwatchlist
sp_regcreate
sp_regnowatch

```

(0 rows affected)

この `isql` の例では、現時点でのレジスタード・プロシージャをすべてリストしています。

#### 使用法

- `sp_reglst` は、現在 Open Server に登録されているプロシージャの名前をすべてロー・データとして返します。
- C 言語プログラムでも、`sp_reglst` を使ってレジスタード・プロシージャをリストできます。

結果は、`SRV_MAXNAME` 文字のデータ長を持つ、1 つの `char` カラムを含んだローとして返されます。

#### 参照

[sp\\_regcreate](#), [sp\\_regdrop](#), [sp\\_regwatch](#), [sp\\_regwatchlist](#)

## sp\_regnowatch

#### 説明

レジスタード・プロシージャの通知 ( ノーティフィケーション ) リストからクライアントを削除します。

#### 構文

```
sp_regnowatch proc_name
```

#### パラメータ

*proc\_name*

レジスタード・プロシージャの名前です。

#### 例

```

dbrpcinit(dbproc, "sp_regnowatch", (DBUSMALLINT)
0);
dbrpcparam(dbproc, "@proc_name", 0, SYBCHAR, 15,
15, "pricechange");
dbrpcsend(dbproc);

```

この例は、`pricechange` レジスタード・プロシージャの通知リストからクライアントを削除します。

#### 使用法

- クライアントが `dbregnowatch` を呼び出すと、このレジスタード・プロシージャは実行します。

- `SRV_C_PROCEXEC` コールバック・ハンドラは、`srv_rpcname` を使って `sp_regnowatch` の実行を確認し、`sp_paramdata` を使って通知要求を削除する対象のプロシージャ名を入手します。

## メッセージ

```
Notification request removed.
```

通知要求は正常に削除されました。

```
proc_name is not a registered procedure.
```

`proc_name` に指定されたプロシージャは、Open Server には登録されていません。

```
No requests pending.
```

クライアントは、プロシージャに対する未処理の通知要求がありませんでした。

```
Unable to remove notification request.
```

Open Server は、通知要求の削除に失敗しました。

## 参照

[sp\\_regcreate](#), [sp\\_regdrop](#), [sp\\_regwatch](#), [sp\\_regnowatch](#), [sp\\_regwatch](#)

## sp\_regwatch

## 説明

レジスタード・プロシージャの通知リストにクライアントを追加します。

## 構文

```
sp_regwatch proc_name [options]
```

## パラメータ

*proc\_name*

クライアントが通知 (ノーティフィケーション) を希望するレジスタード・プロシージャの名前です。

*options*

クライアントへの通知を、一度しか行わないか、またはプロシージャを実行するごとに行うか、および通知が同期かまたは非同期かを指定する `CS_SMALLINT` です。次の表 4-4 では、*options* に設定できる値を示します。これらの値はビット・フラグなので、一度に複数を設定できます。

表 4-4 : `sp_regwatch` の options パラメータの値

option の値	機能
<code>CS_NOTIFY_NOWAIT</code>	非同期の通知を示す。
<code>CS_NOTIFY_WAIT</code>	同期の通知を示す。
<code>SRV_NOTIFY_ALWAYS</code>	Open Server はクライアントが切断するか、 <code>srv_regnowatch</code> か <code>dbregnowatch</code> を呼び出すまでプロシージャを実行するごとにクライアントに通知する。これがデフォルト値です。
<code>SRV_NOTIFY_ONCE</code>	Open Server は通知を送信した後で、通知リストからクライアントを削除する。

例

```
dbrpcinit(dbproc, "sp_regwatch", (DBUSMALLINT) 0);
dbrpcparam(dbproc, "@proc_name", 0, SYBCHAR,
            15, 15, "pricechange");
dbrpcsend(dbproc);
```

この例では、`pricechange` というプロシージャの通知リストにクライアントを追加します。プロシージャが実行されるたびに、クライアントは通知を受けます。

```
optionval = SRV_NOTIFY_ONCE;
dbrpcinit(dbproc, sp_regwatch, (DBUSMALLINT)
          DBWAIT);
dbrpcparam(dbproc, "@proc_name", 0, SYBCHAR,
            15, 15, "pricechange");
dbrpcparam(dbproc, "@options", 0, SYBINT4, -1,
            -1, &optionval);
dbrpcsend(dbproc);
```

この例では、`pricechange` というプロシージャの通知リストにクライアントを追加します。クライアントは、一度だけプロシージャが実行したという通知を受けています。

使用法

- Open Server は、クライアントが `dbnpwatch` を呼び出すと、`sp_regwatch` を内部的に実行します。
- クライアントが通知を待っている間にプロシージャが削除された場合、クライアントは、プロシージャはもはや登録されていないことを示すエラー・メッセージを受け取ります。

メッセージ

```
Notification request added.
```

通知要求が正常に追加されました。

```
proc_name is not a registered procedure.
```

`proc_name` パラメータで指定されたプロシージャは、Open Server では登録されていません。

Unable to add notification request.

Open Server は何らかの理由で要求を追加できませんでした。

参照

[sp\\_regcreate](#), [sp\\_regnowatch](#), [sp\\_regdrop](#)

## sp\_regwatchlist

**説明** クライアントが通知を要求したレジスタード・プロシージャをリストします。

**構文** sp\_regwatchlist

**例**

```
1>execute utility...sp_regwatchlist
2>go
```

```
Procedure Name
-----
pricechange
```

このサーバ間 RPC の isql の例は、クライアントが pricechange レジスタード・プロシージャの通知を要求したことを示しています。

**使用法**

- Open Server は、クライアントが dbregwatchlist を呼び出すと、sp\_regwatchlist を内部的に実行します。
- SRV\_C\_PROCEXEC コールバック・ハンドラは、srv\_rpcname を呼び出し、sp\_regwatchlist が実行しているのを確認することができます。

結果は、SRV\_MAXNAME 文字の 1 つの char カラムを含むローとして返されます。

参照

[sp\\_reglis](#), [sp\\_regwatchlist](#)

## sp\_serverinfo

**説明** クライアントに文字セットまたはソート順の情報を送ります。

**構文** sp\_serverinfo function [name]

**パラメータ** function

表 4-5 は function の有効値を示します。



表 4-5 : function の値 (sp\_serverinfo)

値	意味
<i>server_csname</i>	Open Server アプリケーションの文字セットの名前は、単一カラムの1つの文字ローとしてクライアントに送信される。
<i>server_soname</i>	Open Server アプリケーションのソート順の名前は、単一カラムの1つの文字ローとしてクライアントに送信される。
<i>csdefinition</i>	文字セット定義を含むローがクライアントへ送信される。このローは、タイプが CS_SMALLINT_TYPE、ID が CS_TINYINT_TYPE、文字セット定義が CS_IMAGE_TYPE の3つのカラムで構成されている。
<i>sodefinition</i>	ソート順定義を含むローがクライアントへ送信される。このローは、タイプが CS_SMALLINT_TYPE、ID が CS_TINYINT_TYPE、ソート順定義が CS_IMAGE_TYPE の3つのカラムで構成されている。

**name**

文字セットまたはソート順の名前です。function が *csdefinition* または *sodefinition* に設定された場合にのみ、*name* が要求されます。

**使用法**

- リモート・プロシージャである *sp\_serverinfo* は、標準システム・プロシージャ、たとえば *sp\_who* として自動的に登録され、処理されます。*sp\_serverinfo* が RPC として受信されると、Open Server が自動的に処理します。アプリケーション・コードは必要ありません。
- クライアントが言語要求で *sp\_serverinfo* 要求を送信した場合、正しい応答を送信するには、レジスタード・プロシージャ・ルーチンを使用してこのストアド・プロシージャを実行しなければなりません。
- この情報はローとしてクライアントへ送られます。

## sp\_terminate

**説明**

Open Server のスレッドを終了します。

**構文**

```
sp_terminate spid [, options]
```

**パラメータ**

*spid*

スレッド ID。これは *sp\_who* プロシージャまたは *srv\_thread\_props* の呼び出しで得られます。

## options

スレッドが即時に終了するか、キューイングされた切断イベントによって終了するかを指定します。以前のイベント処理後に発生した切断イベントをキューイングするには、“deferred”を指定してください。これがデフォルトです。スレッドの現在のイベントやキューイングされたイベントを無視してスレッドを即座に終了させるには、“immediate”を指定してください。

## 例

```
1> execute utility...sp_who
2>go
```

spid	status	loginame	hostname	blk	cmd
1	runnable			0	NETWORK HANDLER
2	sleeping			0	CONNECT HANDLER
3	sleeping			0	DEFERRED HANDLER
4	runnable			0	SCHEDULER
12	runnable	ned	sonoma	0	PRINT TASK
24	running	bud	sonoma	0	

(0 rows affected)

この例は、`isql` を使って、異常を起こしたサーバ・スレッドを検索し、終了する方法を示しています。スレッドは即座に終了します。

```
1> execute utility...sp_terminate 12, "immediate"
2>go
```

```
spid = 12;
dbrpcinit(dbproc, "sp_terminate", (DBUSMALLINT) 0);
dbrpcparam(dbproc, "@spid", 0, SYBINT4, -1,
           -1, &spid);
dbrpcparam(dbproc, "@options", 0, SYBCHAR, 9,
           9, "deferred");
dbrpcsend(dbproc);
```

この DB-Library の例では、スレッドを使用してスレッドの `SRV_DISCONNECT` イベントをキューイングしています。次にスレッドが実行可能になった時点で、このスレッドが切断イベントを受信し、終了します。

## 使用法

- `sp_who` または `sp_ps` を使用して、終了の対象となるスレッドの `spid` を検索します。
- Server-Library プログラムでは、`srv_termproc` を使用しスレッドを終了します。

## メッセージ

`spid terminated.` (spid は終了されました)

*spid* scheduled for termination. (*spid* は終了するようにスケジューリングされました)

*spid* not currently in use. (*spid* は現在使用されていません)

参照

[sp\\_who](#), [srv\\_termproc](#)

## sp\_who

説明 指定された Open Server スレッドに関するステータス情報を返します。

構文 `sp_who [loginame | ' spid' ]`

パラメータ *loginame*

ユーザのログイン名です。

*spid*

レポートの対象となるスレッドの内部 ID 番号です。 *spid* は、以前の `sp_ps` または `sp_who` 呼び出しから得られます。 *spid* が指定されていないと、すべてのスレッドがリストされます。

例

```
1> execute utility...sp_who
2>go
```

spid	status	loginame	hostname	blk	cmd
1	runnable			0	NETWORK HANDLER
2	sleeping			0	CONNECT HANDLER
3	sleeping			0	DEFERRED HANDLER
4	runnable			0	SCHEDULER
11	sleeping		hiram	0	
14	running	bud	sonoma	0	

この例は、`sp_who` プロシージャからの出力を示しています。

使用法

- `sp_who` は、指定されたサーバ・スレッド、またはすべての現在の Open Server スレッドについてのステータス情報をレポートします。
- `sp_who` システム・レジスタード・プロシージャからの出力は、Adaptive Server Enterprise の `sp_who` システム・プロシージャの出力と一致します。
- `sp_who` は `sp_ps` が返す情報のサブセットを返します。

- *loginame* および *spid* は文字列パラメータです。Adaptive Server Enterprise からのリモート・プロシージャ・コールとして *isql* を使用し、*sp\_who* を実行するときは、構文エラーを避けるために、*spid* を一重引用符 (') で囲みます。
- *loginame* または *spid* を指定しなければ、*sp\_who* はすべての現在のスレッドをリストします。
- *sp\_who* は次の情報を返します。

*spid* — スレッドの内部スレッド番号。

*status* — スレッドの現在のステータス。このカラムの値は、次のとおりです。

- running
- runnable
- sleeping
- sick
- free
- stopped
- spawned
- terminal
- unknown

1 つしかない “running” タスクは、*sp\_who* を実行しているスレッドです。

*loginame* — ログインしたユーザの名前です。クライアント・スレッドにのみ適用される。

*hostname* — クライアント・タスクの場合には、クライアント・マシンの名前です。サイト・ハンドラ・スレッドの場合には、リモート Adaptive Server Enterprise の名前です。

*blk* — このフィールドは使用されず、常に 0 に設定されています。

*cmd* — スレッドのステータスを記述する文字列です。

*srv\_thread\_props* ルーチンでこのカラムの内容を設定します。

表 4-6 は、各カラムにローの形式で返される結果を示しています。

表 4-6 : 返される情報のフォーマット (sp\_who)

カラム名	データ型	長さ
spid	CS_INT_TYPE	4
status	CS_CHAR_TYPE	10
loginame	CS_CHAR_TYPE	12
hostname	CS_CHAR_TYPE	10
blk	CS_INT_TYPE	3
cmd	CS_CHAR_TYPE	16

参照

[sp\\_ps](#), [sp\\_terminate](#)



# Open Server エラー・メッセージ

Open Server 使用時に発生する可能性のある構成、開発、配備、および運用上の問題を解決します。

Open Server のエラータイプ、重大度、および処理については、「[エラー](#)」(97 ページ) を参照してください。

## エラー 16000

重大度	10
メッセージ	オプション '%!%' が不明です。
説明	srv_reginit() に渡されたオプションが無効です。
アクション	有効なオプションを srv_reginit() に渡してください。「 <a href="#">srv_reginit</a> 」(400 ページ) を参照してください。
追加情報	記号定数 : SRV_EOPTION

## エラー 16001

重大度	10
メッセージ	空きサーバ・プロセスがありません。
説明	サービス・スレッドを割り当てることができません。このエラーはリソースが不十分なために発生することがあります。
アクション	サーバ・ログを確認してください。この問題の基本的な原因を示すメッセージは通常、エラー 16001 よりも前に表示されます。
追加情報	記号定数 : SRV_ENOPROC

## エラー 16002 ( 廃止予定 )

重大度	N/A
メッセージ	アクティブな i/o を使用してクライアント・サーバ・プロセス '%1!' を削除してみてください。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ELIVEPROC

## エラー 16003

重大度	10 または 15
メッセージ	プロセス '%2!' で要求されたイベント '%1!' が不明です。
説明	不明なイベントが <code>srv_event()</code> に渡されました。このエラーはスレッド作成時にも発生することがあります。つまり、新規の <code>SRV_PROC</code> にイベントを割り当てることはできません。
アクション	<code>srv_event()</code> には、以下のユーザ定義イベント <code>SRV_DISCONNECT</code> 、 <code>SRV_URGDISCONNECT</code> 、 <code>SRV_STOP</code> のみを渡してください。
追加情報	記号定数 : SRV_ENOEVENT

## エラー 16004

重大度	10
メッセージ	プロセス '%2!' で要求されたイベント '%1!' が不明です。
説明	<code>srv_event()</code> は、 <code>SRV_DISCONNECT</code> 、 <code>SRV_URGDISCONNECT</code> 、 <code>SRV_STOP</code> 以外のイベント、またはユーザ定義イベントを受信しました。
アクション	<code>srv_event()</code> には、有効なイベントのみを渡してください。
追加情報	記号定数 : SRV_EBADEVENT



## エラー 16005

重大度	10
メッセージ	%2! におけるイベント '%1!' がわかりません。
説明	アプリケーションで、 <code>srv_handle()</code> を使用して無効なイベント・ハンドラをインストールしようとしてしました。
アクション	標準の Open Server イベント ( <code>SRV_CONNECT</code> または <code>SRV_LANGUAGE</code> など)、あるいは <code>srv_define_event()</code> を使用して定義したユーザ定義イベントをインストールしてください。
追加情報	記号定数: <code>SRV_EBADUEVENT</code>

## エラー 16006 ( 廃止予定 )

重大度	N/A
メッセージ	送信バッファ <code>sleft &lt; 0 (%1!)</code> が壊れています。
説明	N/A
アクション	N/A
追加情報	記号定数: <code>SRV_EIOBADBUF</code>

## エラー 16007 ( 廃止予定 )

重大度	N/A
メッセージ	<code>srv__buf_io()</code> で要求された I/O オペレーション (%1!) がわかりません。
説明	N/A
アクション	N/A
追加情報	記号定数: <code>SRV_EIOBADTYPE</code>

## エラー 16008

重大度	10、15、または20
メッセージ	'%1!' バイトのメモリ割り当てが '%3!' の割り当ての '%2!' で異常終了しました。
説明	通常、このエラーは、指定された関数で必要なバイト数が割り当てられない場合に発生します。別の <b>Open Server</b> ルーチンでもこのエラーが発生することがあります。
アクション	カーネルとオペレーティング・システム・パラメータ ( <b>ulimit -a</b> ) によって、 <b>Open Server</b> が必要なメモリを割り当てられることを確認してください。 <b>Open Server</b> のメモリ使用量を監視して、メモリ・リークがないか判別してください。
追加情報	記号定数 : <b>SRV_EMEMORY</b>

## エラー 16009

重大度	20
メッセージ	'%1!' プロセスのサーバ・プロセスの作成ができません。
説明	新規スレッドのスタック割り当て中にメモリ不足が生じたため、 <b>makecontext()</b> が失敗しました。
アクション	カーネルとオペレーティング・システム・パラメータ ( <b>ulimit -a</b> ) によって、 <b>Open Server</b> が必要なメモリを割り当てられることを確認してください。 <b>Open Server</b> のメモリ使用量を監視して、メモリ・リークがないか判別してください。
追加情報	記号定数 : <b>SRV_ENOPPROC</b>

## エラー 16010

重大度	10
メッセージ	マシン '%2!' 上のネットワーク '%1!' の初期化が異常終了しました。
説明	ファイル <i>sql.ini</i> または <i>interfaces</i> に不明のネットワーク・インタフェースが含まれています。

アクション ファイル *sql.ini* および *interfaces* をチェックして、Open Server が有効なエントリを受信していることを確認してください。

追加情報 記号定数: SRV\_EINITNET

## エラー 16011 ( 廃止予定 )

重大度 N/A

メッセージ インタフェース・ファイルが正しくありません。

説明 N/A

アクション N/A

追加情報 記号定数: SRV\_EIFILE

## エラー 16012

重大度 10

メッセージ %1! ファイル '%2!' のオープンができません。

説明 指定されたログ・ファイルを開けません。

アクション 以下を確認してください。

- ファイルが別のプロセスで使用中にはなっていない。
- ファイルのパスと名前が正しい。
- Open Server にこのファイルの読み込みパーミッションが与えられている。

追加情報 記号定数: SRV\_EFOPEN

## エラー 16013

重大度 20

メッセージ インタフェース・ファイル中にサーバ名 '%1!' が見つかりませんでした。

## エラー 16014 ( 廃止予定 )

---

説明	指定されたサーバ名が <i>sql.ini</i> ファイルまたは <i>interfaces</i> ファイルに見つかりません。
アクション	コマンド・ラインで指定された <b>DSL</b> LISTEN 環境変数とサーバ名をチェックしてください。サーバ名の大文字小文字の区別とスペルが <i>sql.ini</i> ファイルまたは <i>interfaces</i> ファイルで指定したとおりであることを確認してください。
追加情報	記号定数 : SRV_ESRVNAME

## エラー 16014 ( 廃止予定 )

重大度	N/A
メッセージ	ユーザ (%1!) のホーム・ディレクトリが見つかりませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EHOMEDIR

## エラー 16015 ( 廃止予定 )

重大度	N/A
メッセージ	ルーチン '%2!' にあるプロセス・タイプ '%1!' がわかりません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EPROCTYPE

## エラー 16016

重大度	15
メッセージ	空きメッセージがありません。

説明	メッセージのキューへの割り当てまたは送信、あるいはキューからのメッセージ取得が実行できません。
アクション	Open Server のトレース・フラグ <code>SRV_TR_MSGQ</code> を使用して、メッセージ・キュー・アクティビティをトレースしてください。必要に応じて、 <code>srv_props(SRV_S_MSGPOOL)</code> プロパティを調整してください。
追加情報	記号定数 : <code>SRV_ENOMSGS</code>

## エラー 16017 ( 廃止予定 )

重大度	N/A
メッセージ	プロセス '%3!' からのメッセージのための %1! プロセス '%2!'。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_EMSGPROC</code>

## エラー 16018 ( 廃止予定 )

重大度	N/A
メッセージ	プロセス '%3!' からのメッセージの %1! プロセス '%2!' メッセージが正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_EMSGADDRESS</code>

## エラー 16019

重大度	10
メッセージ	%1! によって受け取ったメッセージ・タイプが正しくありません。

## エラー 16020 ( 廃止予定 )

---

説明	クライアント・アプリケーションからネゴシエートされたログイン情報を読み込んでいる間に、 <code>srv_negotiate()</code> が不正なメッセージ・タイプを受信しました。
アクション	クライアント・アプリケーションと <b>Open Server</b> の間で <b>Ribo</b> を実行して、クライアントによって送信されたメッセージ・タイプを確認してください。このチェックは <a href="#">表 3-71 (357 ページ)</a> にリストした有効なメッセージ・タイプと照らし合わせて行ってください。
追加情報	記号定数 : <code>SRV_EMSGTYPE</code>

## エラー 16020 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント '%2!' に対する <code>srv__runnet()</code> からのイベント・タイプ '%1!' が正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_EBADIOEVENT</code>

## エラー 16021 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント・プロセス '%3!' で I/O エラー '%1!' (%2!) がありました。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_EIOERROR</code>

## エラー 16022 ( 廃止予定 )

重大度	N/A
メッセージ	%2! でサーバ・プロセス ID '%1!' が正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EQPROC

## エラー 16023 ( 廃止予定 )

重大度	N/A
メッセージ	実行キュー構造体に空きがありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EQFREE

## エラー 16024 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント・プロセス spid %1! が非イベント・メッセージを受け取りました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ESRVEVENT

## エラー 16025 ( 廃止予定 )

重大度	N/A
-----	-----

## エラー 16026 ( 廃止予定 )

---

メッセージ	%2! に与えられたメッセージ・タイプ '%1!' がわかりません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECLNTMSG

## エラー 16026 ( 廃止予定 )

重大度	N/A
メッセージ	受信ソケットの作成ができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ESOCKET

## エラー 16027 ( 廃止予定 )

重大度	N/A
メッセージ	ソケットのバインドができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EBIND

## エラー 16028 ( 廃止予定 )

重大度	N/A
メッセージ	ホスト '%1!' が見つかりません。
説明	N/A
アクション	N/A



追加情報 記号定数: SRV\_EGETHOST

## エラー 16029

重大度 20

メッセージ ネットワーク・リスナの起動ができませんでした。

説明 Open Server は指定されたポート上でクライアント接続を受信できません。

アクション netstat -an を使用して、サーバの *interfaces* エントリに関連付けられているポートが使用中でないことを確認してください。ポートが使用中でない場合は、エラー・ログでその他のエラーがないか確認してください。

追加情報 記号定数: SRV\_ELISTEN

## エラー 16030 ( 廃止予定 )

重大度 N/A

メッセージ クライアント・ソケットの接続ができません。

説明 N/A

アクション N/A

追加情報 記号定数: SRV\_ECNTSOCKET

## エラー 16031 ( 廃止予定 )

重大度 N/A

メッセージ サーバ・プロセス '%1!' は '%2!' でアクティブではありません。

説明 N/A

アクション N/A

追加情報 記号定数: SRV\_EPROCDEAD

## エラー 16032

重大度	10
メッセージ	'%2!' へ渡された STRCHR (%1!) が正しくありません。
説明	このエラーは、異なるデータ形式間での変換 (たとえば、バイト・スワッピングなど) に使用される内部関数のポインタ配列が無効なインデックス値を指定した場合に発生することがあります。
アクション	Open Server に渡された LOGINREC が壊れていないことを確認してください。
追加情報	記号定数 : SRV_EBADINDEX

## エラー 16033

重大度	10
メッセージ	タイプ '%1!' からタイプ '%2!' へ変換することができません。
説明	CS_DATAFMT が <code>srv_bind()</code> または <code>srv_alt_bind()</code> に記述したデータ型に変換できないデータがクライアント・アプリケーションから送信されました。
アクション	サポートされているデータ型のみがクライアント・アプリケーションから送信されるようにしてください。『共通ライブラリ・リファレンス・マニュアル』のデータ型変換チャートを参照してください。
追加情報	記号定数 : SRV_ENOCONVERT

## エラー 16034

重大度	10
メッセージ	%1 名は (%3! があれば) %2! バイトよりも長くなってはいけません。
説明	カラム名が CS_MAX_NAME よりも長くなっています。
アクション	CS_MAX_NAME の現在の定義については、 <code>cstypes.h</code> を参照してください。
追加情報	記号定数 : SRV_EMAXNAME

## エラー 16035 ( 廃止予定 )

重大度	N/A
メッセージ	タイプ '%1!' からタイプ '%2!' への変換がアンダーフローとなりました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECUFLOW

## エラー 16036 ( 廃止予定 )

重大度	N/A
メッセージ	タイプ '%1!' からタイプ '%2!' への変換がオーバーフローとなりました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECOFLOW

## エラー 16037 ( 廃止予定 )

重大度	N/A
メッセージ	バイナリ変数への変換がオーバーフローとなりました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECBINOFLOW

## エラー 16038 ( 廃止予定 )

重大度	N/A
メッセージ	タイプ '%1!' からタイプ '%2!' への変換により総桁数が失われました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECPREC

## エラー 16039 ( 廃止予定 )

重大度	N/A
メッセージ	ソース・フィールドにおける構文エラーによってタイプ '%1!' からタイプ '%2!' への変換が中断されました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECSYNTAX

## エラー 16040 ( 廃止予定 )

重大度	N/A
メッセージ	整数カラム長 '%1!' が正しくありません。正しい値は 1、2、および 4 バイトです。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EBADLEN

## エラー 16041 ( 廃止予定 )

重大度	N/A
メッセージ	可変長データ型 '%1!' が不明です。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EUVDT

## エラー 16042 ( 廃止予定 )

重大度	N/A
メッセージ	固定長データ型 '%1!' が不明です。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EUFDT

## エラー 16043

重大度	10
メッセージ	ロー記述を付けずにローを送ろうとしました。
説明	srv_alt_xferdata()、srv_send_text()、または srv_send_data() が対応するロー記述なしでローを送信しようとしていました。
アクション	srv_alt_xferdata() が実行できなかった場合は、srv_alt_descfmt() を呼び出してローを記述してください。それ以外の場合は、srv_descfmt() を呼び出してください。
追加情報	記号定数 : SRV_ENOROWDESC

## エラー 16044

重大度	10
メッセージ	%1! データ型 '%2!' が既知のデータ型ではありません。
説明	無効なデータ型が CS_DATAFMT 構造または CS_IODESC 構造に定義されています。
アクション	CS_DATAFMT 構造または CS_IODESC 構造のデータ型の有効性を確認してください。
追加情報	記号定数: SRV_ENOTYPE

## エラー 16045

重大度	10
メッセージ	%1!: %2! のデータ項目がありません。
説明	クライアント・アプリケーションから COMPUTE 句を含むクエリが送信されましたが、Open Server では <code>srv_alt_descfmt()</code> を呼び出していません。
アクション	Open Server が <code>srv_alt_descfmt()</code> を呼び出して、クライアント・アプリケーションに対して COMPUTE カラム形式を記述していることを確認してください。
追加情報	記号定数: SRV_ENOITEM

## エラー 16046

重大度	10 または 20。
メッセージ	%1!: 結果ステータス・エラー - %2! はすでにクライアントへ送られています。
説明	この問題は、 <code>srv_negotiate()</code> の呼び出しの間で <code>srv_senddone()</code> を送信せずにネゴシエートされたログイン・ディレクティブが送信された場合、 <code>srv_negotiate()</code> の呼び出し中に発生することがあります。

アクション	<code>srv_negotiate()</code> の呼び出しの間に <code>srv_senddone(SRV_DONE_FINAL)</code> または <code>srv_senddone(SRV_DONE_MORE)</code> を送信してください。また、Open Server アプリケーション・コードをチェックして、 <code>srv_senddone()</code> の最終呼び出しのあとでパラメータが返されていないことを確認してください。
追加情報	記号定数: <code>SRV_ESTATSTATE</code>

## エラー 16047

重大度	10
メッセージ	そのようなトレース・フラグ (%1!) はありません。
説明	<code>srv_props(SRV_S_TRACEFLAG)</code> は無効なトレース・フラグを受け取りました。
アクション	以下のトレース・フラグのみが <code>srv_props(SRV_S_TRACEFLAG)</code> に渡されていることを検証してください。 <code>SRV_TR_TDSHDR</code> 、 <code>SRV_TR_TDSDATA</code> 、 <code>SRV_TR_MSGQ</code> 、 <code>SRV_TR_ATTN</code> 、 <code>SRV_TR_TDSTOKEN</code> 、 <code>SRV_TR_EVENT</code> 、 <code>SRV_TR_NETPARAM</code> 、 <code>SRV_TR_NETREQ</code> 、 <code>SRV_TR_NETDRIVER</code> 、 <code>SRV_TR_NETMEM</code> 、 <code>SRV_TR_NETDATA</code> 、 <code>SRV_TR_NETWAKE</code> 、 <code>SRV_TR_DEFQUEUE</code> 。
追加情報	記号定数: <code>SRV_ENOTRACE</code>

## エラー 16048

重大度	10
メッセージ	カレント・リモート・プロシージャ・コールがありません。
説明	Open Server で <code>SRV_RPC</code> イベント・ハンドラの外側から <code>RPC</code> (リモート・プロシージャ・コール) の処理が試みられましたが、 <code>RPC</code> または <code>RPC</code> パラメータは定義されていません。
アクション	<code>SRV_RPC</code> イベント・ハンドラのみから <code>srv_rpcoptions()</code> を呼び出してください。
追加情報	記号定数: <code>SRV_ENORPC</code>

## エラー 16049

重大度	10
メッセージ	%1!: %3! のデータ項目番号 %2! がありません。
説明	カラム情報またはパラメータ情報を返すルーチン (blk_colval() など) で指定されたカラム番号が無効です。
アクション	テーブル定義に登録されているカラム番号を指定してください。
追加情報	記号定数 : SRV_ENOITEMNO

## エラー 16050

重大度	10
メッセージ	RPC パラメータ番号 %1! がリターン・パラメータではありません。
説明	srv_bind()、srv_descfmt()、または srv_xferdata() の呼出中にリターン・パラメータとして指定されたパラメータは、リターン・パラメータではありません。
アクション	RPC の定義をチェックして、このパラメータがリターン・パラメータであることを確認してください。
追加情報	記号定数 : SRV_ENORETPARAM

## エラー 16051 ( 廃止予定 )

重大度	N/A
メッセージ	タイプ '%2!' の長さ '%1!' が正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EBADPARMLEN



## エラー 16052

重大度	10
メッセージ	実際の長さ '%1!' を最大長 '%2!' よりも大きくすることはできません。
説明	<code>srv_describe()</code> に指定されたカラム名が <code>CS_MAX_NAME</code> よりも長くなっています。
アクション	カラム名の長さが <code>CS_MAX_NAME</code> 以下であることを確認してください。
追加情報	記号定数: <code>SRV_EBADPARAMMAXLEN</code>

## エラー 16053

重大度	10
メッセージ	<code>srv_symbol()</code> へ渡された '%1! '%2!' がわかりません。
説明	<code>srv_symbol()</code> が不明の記号を受け取りました。
アクション	変換可能記号のリストについては、 <a href="#">表 3-139 (474 ページ)</a> を参照してください。
追加情報	記号定数: <code>SRV_ESYMBOL</code>

## エラー 16054

重大度	10
メッセージ	そのようなサーバ・フィールド (%1!) はありません。
説明	<code>srv_sfield()</code> の呼び出しに指定されたフィールドが見つかりません。
アクション	<code>srv_sfield()</code> はユーザが使用するものではありません。Sybase サポート・センタにお問い合わせください。
追加情報	記号定数: <code>SRV_ESRVTYPE</code>

## エラー 16055

重大度	10
メッセージ	そのようなサーバ・プロセス・フィールド (%1!) はありません。
説明	srv_pfield() の呼び出し中に指定されたフィールドが見つかりません。
アクション	srv_pfield() はユーザが使用するものではありません。Sybase サポート・センタに問い合わせてください。
追加情報	記号定数 : SRV_ESRVPROCTYPE

## エラー 16056

重大度	10
メッセージ	%2! へ NULL %1! ポインタが渡されました。
説明	NULL パラメータを受け入れない API で NULL ポインタが受信されました。
アクション	指定した API に渡されたパラメータが NULL でないことを検証してください。
追加情報	記号定数 : SRV_ENULLPTR

## エラー 16057

重大度	10
メッセージ	下位レベル TDS -- ステータス ENVCHANGE またはパラメータを送信しようとしたが無視されました。
説明	クライアント・アプリケーションは、Open Server が送信しようとしたステータス、TDS 環境変更トークン、またはパラメータをサポートしていません。
アクション	Open Server によって送信されたオブジェクトが必須の場合は、クライアント・アプリケーションの TDS (Tabular Data Stream™) レベルを更新してください。
追加情報	記号定数 : SRV_ETDVERSION

## エラー 16058

重大度	20
メッセージ	ネットワーク ATTENTION/OOB ハンドラのインストールができません でした。 -- 致命的エラー
説明	このエラーは廃止され、現在は使用されていません。
アクション	Open Server アプリケーションをサポートされているバージョンに更新 してください。
追加情報	記号定数: SRV_EOOB

## エラー 16059 ( 廃止予定 )

重大度	N/A
メッセージ	TCP Select は SIGURG を送ったソケットがなにもないことを確認し ました。 -- SIGURG が無視されました。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENO_OOB

## エラー 16060 ( 廃止予定 )

重大度	N/A
メッセージ	%1! 状態のために TCP ソケット SIGURG 設定が異常終了しました。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ESIGURG_SETUP

## エラー 16061

重大度	15
メッセージ	スレッド %1! がサイト・ハンドラではありません。type = %2!、function = %3!
説明	Open Server は、指定された関数に対して SRV_SITEHANDLER スレッドを予想していました。
アクション	SRV_T_LOGINTYPE プロパティを使用して、ログイン・プロセス中に受信されたログイン・レコードが SRV_SITEHANDLER であることを検証してください。
追加情報	記号定数 : SRV_ENOTSITE

## エラー 16062

重大度	10
メッセージ	オプション %2! の構成オプション値 (%1!) が正しくありません。
説明	廃止予定コマンド <code>srv_config()</code> が無効なオプションを受信しました。
アクション	<code>srv_config()</code> に渡された構成オプションのデータ型を訂正してください。たとえば、 <code>boolean</code> オプションを設定している場合は、 <code>CS_TRUE</code> または <code>CS_FALSE</code> を渡してください。
追加情報	記号定数 : SRV_EBADCONFIGOPT

## エラー 16063

重大度	10
メッセージ	オブジェクト名の構文エラーです。site.db.owner.object;1 を使用してください。
説明	Open Server は、フォーム <code>site.db.owner.object;1</code> に一致しない RPC を受信しました。
アクション	Open Server トレース・フラグ <code>SRV_TR_EVENT</code> および <code>SRV_TR_TDSDATA</code> を有効にして RPC の形式を確認し、必要な訂正を加えてください。

追加情報 記号定数: SRV\_EOBSYNTAX

## エラー 16064 ( 廃止予定 )

重大度 N/A  
メッセージ ネットワークの終了に失敗しました。  
説明 N/A  
アクション N/A  
追加情報 記号定数: SRV\_ENETTERM

## エラー 16065

重大度 10  
メッセージ %! : 無効ステータス - %2!  
説明 構造に無効なステータスが割り当てられています。  
アクション 一部の構造に *status* フィールドが含まれています。有効なステータス値については、以下を参照してください。

- 表 2-9 ( 60 ページ ) - CS\_DATAFMT 構造の場合
- 表 2-10 ( 66 ページ ) - CS\_SERVERMSG の場合
- 表 2-13 ( 74 ページ ) - SRV\_CURDESC の場合

追加情報 記号定数: SRV\_EBADSTATUS

## エラー 16066 ( 廃止予定 )

重大度 N/A  
メッセージ カレントのクライアント・コマンドは RPC です。パラメータを返すには、`srv__paramset()` を使用してパラメータを返してください。  
説明 N/A

アクション	N/A
追加情報	記号定数 : SRV_ERETVALRPC

## エラー 16067

重大度	10
メッセージ	データ型 '%2!' の長さ (%1!) が正しくありません。
説明	指定されたデータ型の長さが無効です。
アクション	Open Server データ型および Open Client データ型の説明については、 <a href="#">表 2-37 (229 ページ)</a> を参照してください。
追加情報	記号定数 : SRV_EBADTYPELEN

## エラー 16068 ( 廃止予定 )

重大度	N/A
メッセージ	現在空のスレッドを削除しようとしています。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EPROCFREE

## エラー 16069 ( 廃止予定 )

重大度	N/A
メッセージ	%1! にある正しくないチャネルのメッセージを読み取ってください。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EINVCHAN

## エラー 16070 ( 廃止予定 )

重大度	N/A
メッセージ	オープンしているサーバ・ログ・ファイルがありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOLOGFILE

## エラー 16071 ( 廃止予定 )

重大度	N/A
メッセージ	カラム番号は %1! よりも大きくなければなりません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECOLNUMBER

## エラー 16072

重大度	10
メッセージ	%1!: 項目番号 '%2!' はすでに記述されています。
説明	同じカラムまたは同じパラメータの情報を取得するために、 <code>srv_descfmt()</code> が複数回呼び出されました。
アクション	<code>srv_descfmt()</code> に渡されたパラメータまたはカラム名が正しいことを確認してください。
追加情報	記号定数 : SRV_EDUPITEM

## エラー 16073

重大度	10
メッセージ	%3! の %1! %2! の記述が完全ではありません。
説明	srv_send_data()、srv_alt_xferdata()、または srv_xferdata() によって、送信中のローで一部のカラムに不完全な記述が検出されました。
アクション	Open Server から送信されるすべてのカラムが完全に記述されていることを確認してください。
追加情報	記号定数 : SRV_EINCDESC

## エラー 16074

重大度	10
メッセージ	ユーザ・イベント (%1!) の構成最大数がすでに定義されています。
説明	srv_define_event() は、新規イベントの追加によってサーバに設定されている最大イベント数が超過するため、実行できませんでした。
アクション	srv_props() を使用して SRV_S_NUMUSEREVENTS を増加してください。
追加情報	記号定数 : SRV_EMAXUEVENT

## エラー 16075

重大度	10
メッセージ	イベント・タイプ (%1!) が正しくありません。
説明	無効なイベント・ハンドラが srv_event() を呼び出したか、無効なイベント・タイプが srv_define_event() に渡されました。
アクション	無効なイベント・ハンドラが原因の場合は、有効イベントのリストについて表 3-76 (365 ページ) を参照してください。無効なイベント・タイプが原因の場合は、SRV_QUEUED を渡してください。これは、現時点では srv_define_event() にとって唯一有効なイベント・タイプです。
追加情報	記号定数 : SRV_EEVENTTYPE



## エラー 16076

重大度	10
メッセージ	アテンション・ハンドラから %1! を呼び出すことはできません。
説明	Open Server は、指定された関数を SRV_ATTENTION ハンドラから呼び出しました。
アクション	次のコマンドのみ (一定の制限付き) をアテンション・ハンドラから呼び出してください。srv_wakeup()、srv_ucwakeup()、srv_thread_props()、srv_props()、および srv_event_deferred()。[アテンション・イベント] (20 ページ) を参照してください。
追加情報	記号定数: SRV_EINATTN

## エラー 16077 ( 廃止予定 )

重大度	N/A
メッセージ	SRV_CONNECT ハンドラから %1! を呼び出すことはできません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_EINCONNECT

## エラー 16078 ( 廃止予定 )

重大度	N/A
メッセージ	サイト・ハンドラ '%1!' からブロックした読み取りを受け取りました。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_EBLOCKREAD

## エラー 16079 ( 廃止予定 )

重大度	N/A
メッセージ	ネットワーク I/O 割り込みハンドラのインストールができませんでした。 -- 致命的エラー
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ESIGIO

## エラー 16080

重大度	20
メッセージ	遅延起動キューがいっぱいです。 SRV_S_DEFQUEUEUSIZE サーバ特性を使用してサイズを増やしてください。
説明	Open Server は <code>srv_deferred_event()</code> を使用して、フルの遅延イベント・キューにイベントを追加しようとしています。このエラーは、Open Server スケジューラの実行時間が十分に長くないためにキュー内のすべての遅延イベントを処理できない場合にも発生することがあります。
アクション	SRV_S_DEFQUEUEUSIZE プロパティの値を増やしてください。スリープしなかったり、ネットワーク I/O を実行しない処理時間の長いルーチンが存在する場合は、このルーチンによってサーバが独占されないように定期的に <code>srv_yield()</code> を呼び出してください。
追加情報	記号定数 : SRV_EDEFQFULL

## エラー 16081 ( 廃止予定 )

重大度	N/A
メッセージ	警告 : Sybase の インストール・ハンドラがこのシグナルの処理のために存在するので、 シグナル %2! の %1! は無視されました。
説明	N/A
アクション	N/A

追加情報 記号定数 : SRV\_EFLAGIGN

## エラー 16082 ( 廃止予定 )

重大度 N/A

メッセージ 同じシグナルに対する Sybase のインストール・ハンドルによって、  
%1! のシグナル・ハンドラが 実行されます。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_ESIGSUPERCEDE

## エラー 16083 ( 廃止予定 )

重大度 N/A

メッセージ 未使用エラーが定義されています。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_EDEFUNUSED

## エラー 16084 ( 廃止予定 )

重大度 N/A

メッセージ サーバを削除中に %2! のタスク・タイプ %1! がわかりません。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_EBADPROCTYPE

## エラー 16085 ( 廃止予定 )

重大度	N/A
メッセージ	DECnet のためのネットワーク制御構造体のメモリ割り当てができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNAALLFAIL

## エラー 16086 ( 廃止予定 )

重大度	N/A
メッセージ	DECnet リスナのメールボックスの作成ができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNACRELSNMBX

## エラー 16087 ( 廃止予定 )

重大度	N/A
メッセージ	DECnet のための _NET への割り当てが異常終了しました。ネットワークを開始することができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNANETASSIGN

## エラー 16088 ( 廃止予定 )

重大度	N/A
メッセージ	ポート名 %1! はすでに DECnet で使用されています。ネットワークを開始することができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNADUPOBJ

## エラー 16089 ( 廃止予定 )

重大度	N/A
メッセージ	DECnet のためのオブジェクト %1! のオブジェクト宣言が異常終了しました。ネットワークを開始することができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNADECLARE

## エラー 16090 ( 廃止予定 )

重大度	N/A
メッセージ	DECnet での受信のための読み取りが異常終了しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNALISTEN

## エラー 16091 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント接続の新しいチャンネルの割り当てができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNAASSIGN

## エラー 16092 ( 廃止予定 )

重大度	N/A
メッセージ	新しいクライアント接続を完了することができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNAACCESS

## エラー 16093 ( 廃止予定 )

重大度	N/A
メッセージ	クライアントで使用可能なリンク・テーブル・エントリがありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDNANOLINK

## エラー 16094

重大度	10
-----	----

メッセージ	サービスを行っていないプロセス %1! を削除しようとしています。
説明	srv_dropproc() (廃止予定コマンド) が SRV_TSERVICE 以外のスレッド・タイプを削除しようとしてしました。
アクション	srv_dropproc() ではなく、srv_termproc() を使用してください。スレッド・タイプのリストについては、表 2-33 (184 ページ) を参照してください。
追加情報	記号定数 : SRV_ENOTSERVICE

## エラー 16095 (廃止予定)

重大度	N/A
メッセージ	プロセス %1! からサービス・プロセス %2! へ非ユーザ定義イベントを送ろうとします。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EEVENTNOSERVICE

## エラー 16096 (廃止予定)

重大度	N/A
メッセージ	プロセス %1! に対するエラー処理遅延イベント通知です。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDEFHANDLER

## エラー 16097

重大度	10
-----	----

## エラー 16098 ( 廃止予定 )

---

メッセージ	RPC パラメータはすべて名前を指定するか、または、位置を指定するか のいずれかによって渡されなければなりません。
説明	このエラーは、1つのパラメータに名前があり、他のパラメータに名 前がない場合に <code>srv_numparams()</code> を呼び出すと発生します。
アクション	パラメータを定義する際、すべてのパラメータに名前を付けるか、ど のパラメータにも名前を付けないようにしてください。
追加情報	記号定数 : <code>SRV_ERPCINCONSIST</code>

## エラー 16098 ( 廃止予定 )

重大度	N/A
メッセージ	プロセスの割り当てができなかったために、サブチャンネル設定が異常終 了しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_ECHANSETUP</code>

## エラー 16099 ( 廃止予定 )

重大度	N/A
メッセージ	スケジューラのスレッドの作成ができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_ENOSCHEDPROC</code>

## エラー 16100

重大度	20
-----	----



メッセージ	遅延割り込みハンドラ・メッセージ・キューの srv__rundef() における障害です。status = %1!
説明	Open Server の srv_getmsgq(SRV_M_WAIT) へ内部呼び出しが失敗しました。
アクション	失敗の原因に関して <i>infop</i> パラメータをチェックし、修正してください。表 3-56 (329 ページ) に、 <i>infop</i> 値の一覧を示します。
追加情報	記号定数: SRV_EDEFERFAIL

## エラー 16101 ( 廃止予定 )

重大度	N/A
メッセージ	srv__rundef() において遅延割り込みハンドラによって受け取った メッセージが正しくありません。 type = %1!、from = %2!
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_EDEFERINVMMSG

## エラー 16102 ( 廃止予定 )

重大度	N/A
メッセージ	サイト・ハンドラによって受け取ったイベント・タイプが正しくありま せん。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_EINVEVENT

## エラー 16103

重大度	10
-----	----

メッセージ	<code>srv_spawn()</code> へ渡されたスタック・サイズ <code>%1!</code> が正しくありません。
説明	<code>srv_spawn()</code> に渡されたスタック・サイズが小さすぎます。
アクション	<code>stacksize</code> パラメータの値が少なくとも <code>RV_MINSTACKSIZE</code> の値と等しいことを確認してください。
追加情報	記号定数 : <code>SRV_EINVSTACK</code>

## エラー 16104

重大度	10 または 20
メッセージ	新しいスレッドにサイズ <code>%1!</code> のスタックを割り当てるできません。
説明	<code>srv_spawn()</code> に渡されたスタック・サイズが大きすぎます。
アクション	<code>stacksize</code> パラメータの値が、 <code>malloc</code> を呼び出して割り当て可能なメモリの量を示す最大スタック・サイズ以下であることを確認してください。
追加情報	記号定数 : <code>SRV_ESTACKALLOC</code>

## エラー 16105

重大度	10
メッセージ	<code>%2!</code> へ渡された優先度 <code>%1!</code> が正しくありません。正しい優先度は <code>%3!</code> から <code>%4!</code> までです。
説明	<code>srv_spawn()</code> または <code>srv_setpri()</code> が無効な <code>priority</code> 値を受け取りました。
アクション	<code>SRV_C_LOWPRIORITY</code> と <code>SRV_C_MAXPRIORITY</code> の間の <code>priority</code> 値を渡してください。
追加情報	記号定数 : <code>SRV_EINVPRIORITY</code>

## エラー 16106

重大度	10
メッセージ	空のプロセス・キューのメモリを割り当てるできませんでした。

説明	Open Server は、セキュリティ・メカニズム名の抽出中にメモリ不足を起しました。
アクション	Open Server アプリケーションを実行しているユーザに対して、メモリ割り当ての制限を増やすか、削除してください。
追加情報	記号定数 : SRV_EALLOCFAIL

## エラー 16107 ( 廃止予定 )

重大度	N/A
メッセージ	プロトコル確認メッセージを割り当てることができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOPRACK

## エラー 16108

重大度	10
メッセージ	%1! が NULL %2! パラメータで呼び出されました。
説明	非 NULL パラメータを予想しているプロシージャが NULL パラメータを受け取りました。
アクション	エラー・メッセージで特定されたパラメータが NULL でないことを確認してください。
追加情報	記号定数 : SRV_ENULLPARAM

## エラー 16109

重大度	10
メッセージ	%2! が正常に返されたときにかぎり %1! が呼び出されます。

説明	指定されたプロシージャは、 <code>srv_regcreate()</code> 、 <code>srv_regdefine()</code> 、または <code>srv_reginit()</code> が完了する前に呼び出されています。
アクション	指定したプロシージャが適切な初期化コマンドが完了するまで呼び出されないように、アプリケーションを作成してください。
追加情報	記号定数 : <code>SRV_EPMSTAT</code>

## エラー 16110

重大度	10
メッセージ	%1! を呼び出した %2! パラメータが正しくありません。
説明	Open Server API に渡されたパラメータの値が無効です。
アクション	Open Server API に渡される多数のパラメータは、特定の値または特定の範囲に限定されています。指定した API に渡されるパラメータが特定の範囲または値に該当することを確認してください。
追加情報	記号定数 : <code>SRV_EBADPARAM</code>

## エラー 16111

重大度	10
メッセージ	%1! へ渡された <code>SRV_PROC</code> はクライアント・スレッドではありません。
説明	<code>srv_regnowatch()</code> 、 <code>srv_regwatch()</code> 、 <code>srv_regwatchlist()</code> 、または <code>srv_sendinfo()</code> は <code>SRV_TCLIENT</code> 以外の <code>SRV_PROC</code> パラメータを受け取りました。
アクション	<code>srv_thread_props(SRV_T_TYPE)</code> を使用して、 <code>SRV_PROC</code> スレッド・タイプを取得してください。スレッド・タイプは <code>SRV_TCLIENT</code> である必要があります。
追加情報	記号定数 : <code>SRV_EPNOTCLIENT</code>

## エラー 16112

重大度	10
メッセージ	%1! で見つかったオブジェクト名が正しくありません。
説明	メッセージ・キューまたはミューテックスを操作する API が無効な長さのオブジェクト名を受け取りました。
アクション	メッセージ・キューまたはミューテックスの名前の長さがゼロより大きく、SRV_MAXNAME よりも小さくなるように、名前を変更してください。
追加情報	記号定数: SRV_EOBJNAME

## エラー 16113

重大度	10
メッセージ	%1! で見つかったオブジェクト ID が正しくありません。
説明	指定されたメッセージ・キューまたはミューテックスの ID が見つかりません。
アクション	正しいオブジェクト ID を判別するには、 <code>srv_getobjid()</code> を呼び出し、メッセージ・キューまたはミューテックスの名前をパラメータとして渡します。
追加情報	記号定数: SRV_EOBJID

## エラー 16114 ( 廃止予定 )

重大度	N/A
メッセージ	%1! で見つかったロック・モードが正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ELOCKMODE

## エラー 16115

重大度	10
メッセージ	スレッドを開始することができませんでした。
説明	srv_createproc() または srv_spawn() を呼び出したため、Open Server は設定値よりも多くのスレッドを作成しました。
アクション	srv_props(SRV_S_NUMTHREADS) を呼び出して、SRV_DEF_NUMTHREADS によって定義されたデフォルト値よりも多くのスレッドが許容されるように Open Server を設定してください。
追加情報	記号定数 : SRV_ENOSPAWN

## エラー 16116 ( 廃止予定 )

重大度	N/A
メッセージ	停止されていなかったスレッド %1! を再び開始しようとしてしました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EINRESTART

## エラー 16117 ( 廃止予定 )

重大度	N/A
メッセージ	実行中のスレッド %1! を停止しようとしてしました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EINSTOP

## エラー 16118

重大度	10
メッセージ	スレッド %1! によって所有されていない mutex のロックを解除しようとしてしました。
説明	srv_unlockmutex() を使用してミューテックスをロック解除しようとしているスレッドは、そのミューテックスの現在の所有者ではありません。
アクション	各スレッドが作成したミューテックスを調べてください。現在、Open Server で指定されたミューテックスのロックまたはロック解除が許可されるスレッドは 1 つだけです。
追加情報	記号定数 : SRV_ENOTOWNER

## エラー 16119

重大度	10
メッセージ	プロシージャ %1! で受け取ったパラメータが多すぎます。
説明	srv_regexec() は、渡されたパラメータ数と定義されたパラメータ数が一致していないことを検出しました。
アクション	登録されたプロシージャの定義を確認して、定義済パラメータ数とクライアント・アプリケーションから渡されるパラメータ数を比較してください。
追加情報	記号定数 : SRV_EPRNUM

## エラー 16120

重大度	10
メッセージ	プロシージャ %2! のパラメータ %1! の名前を与えなくてはなりません。名前は他のパラメータで使用されたことがあります。
説明	クライアント・アプリケーションは、最初に名前付きパラメータを渡した後で、名前のないパラメータを渡しました。パラメータを定義する際、すべてのパラメータに名前を付けるか、どのパラメータにも名前を付けないようにしてください。

アクション	クライアント・アプリケーションと Open Server 間で Ribo を実行して、名前付きパラメータと名前なしパラメータを識別します。クライアント・アプリケーションを適宜訂正してください。
追加情報	記号定数 : SRV_EPARAMNAME

## エラー 16121

重大度	10
メッセージ	プロシージャ %2! のパラメータ %1! のデータ型が正しくありませんでした。
説明	クライアント・アプリケーションによって、ストアド・プロシージャの定義と一致しないパラメータ・データ型が定義されています。
アクション	クライアント・アプリケーションのパラメータ・データ型の定義がストアド・プロシージャのものと一致することを確認してください。
追加情報	記号定数 : SRV_EPTYPE

## エラー 16122

重大度	10
メッセージ	わからないプロシージャ %1! の登録を解除しようとしてしました。
説明	Open Server は、 <code>srv_regdrop()</code> の呼び出し中に指定されたプロシージャを見つけられませんでした。
アクション	詳細については、 <code>srv_regdrop()</code> * <i>infop</i> パラメータを確認してください。
追加情報	記号定数 : SRV_EPUNREGNOPROC.

## エラー 16123

重大度	10
メッセージ	プロシージャ %1! を実行しようとしてしました。このプロシージャは登録されませんでした。



説明	Open Server は、登録されていないレジスタード・プロシージャを実行しようとした。
アクション	詳細については、 <code>srv_regexec()</code> <i>*infop</i> パラメータを確認してください。
追加情報	記号定数 : <code>SRV_EPROCNOTREG</code>

## エラー 16124

重大度	10
メッセージ	%1! %2! はすでにあるため作成できません。
説明	Open Server は、すでに存在している名前付きミューテックス、メッセージ・キュー、またはレジスタード・プロシージャを作成しようとした。
アクション	作成したミューテックス、メッセージ・キュー、またはレジスタード・プロシージャにユニークな名前を割り当ててください。
追加情報	記号定数 : <code>SRV_EPEXISTS</code>

## エラー 16125

重大度	10
メッセージ	不明なプロシージャ %1! のノーティフィケーションがリクエストされました。
説明	<code>srv_regwatch()</code> が受け取ったプロシージャは見つかりませんでした。
アクション	<code>srv_regwatch()</code> に渡されたプロシージャ名が Open Server にあり、プロシージャ名定義の大文字小文字の区別と一致していることを確認してください。
追加情報	記号定数 : <code>SRV_EPNOPROC</code>

## エラー 16126

重大度	10
メッセージ	クライアントはプロシージャ %1! のノーティフィケーション・リクエストをもっていません。
説明	srv_regnowatch() は、指定されたプロシージャからの通知を待機しているスレッドを見つけることができません。
アクション	詳細については、srv_regnowatch() * <i>infop</i> パラメータを確認してください。
追加情報	記号定数 : SRV_EPNONOTICEREQ

## エラー 16127

重大度	10
メッセージ	SRV_PROC を割り当ててシステム・プロシージャを登録することができませんでした。
説明	システムにより登録されたストアド・プロシージャの作成中または削除中に、Open Server は新しい SRV_TSERVICE スレッドを割り当てることができません。
アクション	このエラーの基本的な原因を説明している可能性のある前出のエラー・メッセージを検索してください。1つの解決法として、Open Server で使用可能なスレッドの最大数を増やすように SRV_S_NUMTHREADS を更新できます。
追加情報	記号定数 : SRV_EPNOSRVPROC

## エラー 16128

重大度	10
メッセージ	プロシージャ %2! で %1! 値を保持するにはバッファ・サイズが不十分です。

説明	<code>srv_dynamic()</code> や <code>srv_options()</code> などの Open Server ルーチンは、バッファ・ポインタ・パラメータを通して情報を返します。指定されたルーチンに渡されるバッファの大きさが十分でない場合、このエラーが生成されます。このエラーは、 <i>*valuep</i> バッファが小さすぎる場合に <code>blk_colval()</code> を呼び出しても発生することがあります。
アクション	適切なバッファ・サイズについては、表 3-42 (307 ページ) ( <code>srv_dynamic()</code> の場合) または表 3-80 (370 ページ) ( <code>srv_options()</code> の場合) を参照してください。
追加情報	記号定数: <code>SRV_EBUFFSIZE</code>

## エラー 16129

重大度	10
メッセージ	プロシージャ %1! がいないため、このプロシージャのノーティフィケーション・リクエストを取り消すことができません。
説明	<code>srv_regnowatch()</code> は、指定されたプロシージャを見つけられません。
アクション	<code>srv_regnowatch()</code> に渡されたプロシージャ名が登録されていて、そのスベルと大文字小文字の区別が正しいことを検証してください。
追加情報	記号定数: <code>SRV_EPNOPROCEX</code>

## エラー 16130

重大度	10
メッセージ	プロシージャ %1! が必要とするパラメータ %2! が与えられていません。
説明	Open server は、クライアント・アプリケーションによって行われた RPC 呼び出しでストアド・プロシージャで必要とするパラメータを見つけることができません。
アクション	プロシージャを実行しているクライアント・アプリケーションが必須パラメータをすべて指定したことを検証してください。クライアント・アプリケーションと Open Server 間で渡されたパラメータを調べるには、 <code>Ribo</code> を実行します。
追加情報	記号定数: <code>SRV_ENEEDPARAM</code>

## エラー 16131

重大度	10
メッセージ	名前をもつパラメータ %1! はレジスタード・プロシージャ %2! にはありません。
説明	クライアント・アプリケーションによって指定したストアド・プロシージャに渡された名前をもつパラメータが見つかりません。
アクション	クライアント・アプリケーションと <b>Open Server</b> 間で渡されたパラメータを調べるには、 <b>Ribo</b> を実行します。
追加情報	記号定数 : <b>SRV_EUNKNOWNPARAM</b>

## エラー 16132

重大度	10
メッセージ	現在の言語コマンドがありません。
説明	<code>srv_langcpy()</code> または <code>srv_langlen()</code> が、イベント・ハンドラ <b>SRV_LANGUAGE</b> または <b>SRV_CURSOR</b> の外側で呼び出されました。
アクション	<code>srv_langcpy()</code> または <code>srv_langlen()</code> は、イベント・ハンドラ <b>SRV_LANGUAGE</b> または <b>SRV_SURSOR</b> の内側のみで呼び出してください。
追加情報	記号定数 : <b>SRV_ENOLANG</b>

## エラー 16133

重大度	10
メッセージ	%1! 接続の設定で限界を超えました。 接続は拒否されました。
説明	最後のクライアント接続要求が、 <b>Open Server</b> で受け入れるように設定された物理的なネットワーク接続の最大数を超えました。
アクション	<b>Open Server</b> プロパティ <b>SRV_S_NUMCONNECTIONS</b> の値を上げて、さらに多くのクライアント接続を受容できるようにしてください。デフォルトは <b>SRV_DEF_NUMCONNECTIONS</b> です。

追加情報 記号定数 : SRV\_EMAXCONNECT

## エラー 16134

重大度 10

メッセージ %1! のコールバック・ルーチンが現在のオペレーションをアボートしました。

説明 タイプ SRV\_C\_PROCEXEC のコールバック・ルーチンが SRV\_S\_INHIBIT を返しました。

アクション 「レジスタード・プロシージャでのコールバック・ハンドラの使用」(190 ページ) を参照してください。

追加情報 記号定数 : SRV\_EPCALLBACK

## エラー 16135

重大度 10

メッセージ プロシージャ %1! のパラメータの最大数はすでに定義されています。

説明 レジスタード・プロシージャは、最大 1024 のパラメータを持つことができます。srv\_regparam() を使用してパラメータ記述を試みましたが、この数字を超えました。

アクション プロシージャのパラメータ数を 1024 に制限してください。

追加情報 記号定数 : SRV\_EMAXPARAM

## エラー 16136

重大度 10

メッセージ ルーチン %1! へ渡されたフラグ値が正しくありません。

説明	<code>srv_getmsgq()</code> 、 <code>srv_putmsgq()</code> 、 <code>srv_lockmutex()</code> 、または <code>srv_wakeup()</code> が無効なフラグを受け取りました。このエラーは、 <code>srv_getobjname()</code> が無効なオブジェクト・タイプを受け取ったときにも発生することがあります。
アクション	現時点で有効なオブジェクト・タイプは <code>SRV_C_MUTEX</code> および <code>SRV_C_QUEUEF</code> のみです。有効なフラグのリストについては、以下を参照してください。 <ul style="list-style-type: none"><li>• 表 3-53 (327 ページ) - <code>srv_getmsgq()</code> の場合</li><li>• 表 3-88 (385 ページ) - <code>srv_putmsgq()</code> の場合</li><li>• 「<code>srv_lockmutex</code>」(344 ページ) - <code>srv_lockmutex()</code> の場合</li><li>• 表 3-153 (498 ページ) - <code>srv_wakeup()</code> の場合</li></ul>
追加情報	記号定数 : <code>SRV_EBADFLAG</code>

## エラー 16137

重大度	10
メッセージ	サイト・ハンドラの最大数 %1! をすでに超えています。
説明	リモート・サーバのサイト・ハンドラからの接続の試みによって、現在設定されている最大数が超過しました。
アクション	<code>SRV_S_NUMREMSITES</code> を更新して、アクティブなリモート・サーバによるサイト・ハンドラ接続の最大数を増やしてください。デフォルトは <code>SRV_DEF_NUMREMSITES</code> です。
追加情報	記号定数 : <code>SRV_EMAXREMSITES</code>

## エラー 16138

重大度	10
メッセージ	このクライアントにはすでにプロシージャ %1! のノーティフィケーション・リクエストがあります。

説明	Open Server は、 <code>srv_regwatch()</code> を使用して、ノーティフィケーション・リストにすでに存在するレジスタード・プロセスに対してスレッドをリストに追加しようしました。
アクション	スレッドはすでにノーティフィケーション・リストに存在するため、アクションをとる必要はありません。
追加情報	記号定数 : <code>SRV_EPNOTIFYEXISTS</code>

## エラー 16139 ( 廃止予定 )

重大度	N/A
メッセージ	リモート・チャンネルの最大数 %1! を超えています。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_EMAXREMCONNS</code>

## エラー 16140

重大度	10
メッセージ	RPC 実行スタックの最大の深さにすでに達しています。
説明	現時点で、再帰的に呼び出し可能な RPC の最大数は 16 です。
アクション	Open Server が再帰的に RPC を呼び出す回数を 16 に制限してください。
追加情報	記号定数 : <code>SRV_EMAXRPC</code>

## エラー 16141

重大度	10
メッセージ	'%1!' へ渡されたサーバ・プロセスは NULL です。
説明	指定された関数は NULL <code>SRV_PROC</code> を受け取りました。

アクション 有効な、NULL 以外の SRV\_PROC を指定された API に渡してください。  
追加情報 記号定数 : SRV\_EPROCNULL

## エラー 16142

重大度 10  
メッセージ '%2!' にサーバ・プロセス・アドレス 0x%1! がプールされていません。  
説明 SRV\_PROC アドレスは、SRV\_SERVER 構造体で管理されているサーバ・プロセスの配列に存在しません。  
アクション SRV\_CURPROC() マクロを使用して、アクティブ・スレッドのアドレスを検索してください。  
追加情報 記号定数 : SRV\_EPROCINVALID

## エラー 16143

重大度 10  
メッセージ サーバ・プロセス・アドレス 0x%1! が '%2!' で正しい境界にありません。  
説明 SRV\_PROC アドレスは SRV\_SERVER 構造体内で管理されるサーバ・プロセスの配列の中に存在しますが、有効範囲の外側に配置されています。  
アクション SRV\_CURPROC() マクロを使用して、現在アクティブなスレッドのアドレスを検索してください。  
追加情報 記号定数 : SRV\_EPROCBOUND

## エラー 16144

重大度 10  
メッセージ '%2!' にサーバ・プロセス '%1!' が割り当てられていません。



説明	このエラーは、 <code>srv_termproc()</code> が Open Server ルーチンに渡された <code>SRV_PROC</code> アドレスをすでに終了していることを示します。
アクション	次のいずれかを実行してください。 <ul style="list-style-type: none"> <li>別のイベントによって Open Server がスレッドを終了したことを示すその他のエラーがないか検索します。</li> <li>終了しようとしている <code>SRV_PROC</code> アドレスを記録する <code>srv_termproc()</code> に関わるコードを作成します。記録されたアドレスを、指定された Open Server ルーチンに渡された <code>SRV_PROC</code> のアドレスと比較します。<code>srv_termproc()</code> が呼び出されない場合は、<code>srv_createproc()</code> が正常に完了していて、Open Server スレッドが完全に初期化されていることを確認します。</li> <li><code>srv_termproc()</code> が割り込みレベル・コードから呼び出されていないことを確認してください。この呼び出しが実行されると、予測不能な結果が生じることがあるためです。</li> </ul>
追加情報	記号定数: <code>SRV_EPROCNOTUSED</code>

## エラー 16145

重大度	10
メッセージ	サーバ・プロセス '%!1!' の '%2!' での I/O が正しくありません。
説明	<code>SRV_PROC</code> の I/O は、たとえば、サーバのシャットダウン時、エラー発生時、プロセスの終了時などにさまざまな理由で無効と見なされることがあります。
アクション	前出のエラー・メッセージを分析して、エラーの基本的な原因を把握してください。スレッドの I/O チャンネルが有効かどうかを判別するには、 <code>srv_thread_props(SRV_T_IODEAD)</code> を使用します。
追加情報	記号定数: <code>SRV_EPROCIODEAD</code>

## エラー 16146 ( 廃止予定 )

重大度	N/A
メッセージ	接続サービス・プロセスが異常終了しています。

説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECONSRVFAIL

## エラー 16147

重大度	20
メッセージ	スレッド '%1!' のネットワーク・バッファを割り当てることができませんでした。
説明	指定されたスレッドによって、Open Server には設定値よりも多くのネットワーク・バッファが割り当てられました。Open Server は、SRV_S_NUMCONNECTIONS、SRV_S_NUMREMBUF、SRV_S_NUMREMSITES などの設定パラメータに基づいてネットワーク・バッファの最大数を割り当てます。
アクション	SRV_T_TYPE プロパティを確認して、どのタイプのスレッドが問題を起こしたか判別してください。適切なプロパティを増やしてください。
追加情報	記号定数 : SRV_ENONETBUF

## エラー 16148

重大度	10
メッセージ	'%1!' が正しい機能値ではありません。
説明	srv_capability() が無効な機能値を受け取りました。
アクション	有効な機能値については、 <a href="#">表 3-19 (277 ページ)</a> を参照してください。
追加情報	記号定数 : SRV_ENOTSUPPORTED

## エラー 16149

重大度	20
-----	----

メッセージ	致命的なエラー：内部 %1! 構造体が %2! では壊れています。
説明	srv_getloginfo() は CS_LOGINFO パラメータを初期化できません。
アクション	srv_getloginfo() を呼び出す前に、SRV_PROC 構造体が有効であることを確認してください。srv_getloginfo() は、SRV_PROC から読み込んだデータを使って CS_LOGINFO パラメータを初期化します。
追加情報	記号定数 : SRV_ECORRUPT

## エラー 16150

重大度	10
メッセージ	スレッド '%1!' が %2! バイトの割り当てスタック・スペースを超える危険があります。
説明	指定されたスレッドは、割り付けられたスタック領域を超える可能性があります。
アクション	SRV_S_STACKSIZE プロパティを増やして、各スレッドのスタックをさらに多く割り当てられるようにしてください。
追加情報	記号定数 : SRV_ELOWSTK

## エラー 16151

重大度	20
メッセージ	スレッド '%1!' のスタックがオーバーフローしているか、または、壊れています。-- アボートします。
説明	指定されたスレッドがそのスタック領域を超えているか、別のスレッドによってスタック領域が上書きされました。
アクション	SRV_S_STACKSIZE プロパティを増やして、各スレッドのスタックをさらに多く割り当てられるようにしてください。
追加情報	記号定数 : SRV_EBADSTK

## エラー 16152 ( 廃止予定 )

重大度	N/A
メッセージ	致命的なエラー：ネットワーク i/o 中にエラーが見つかりました。エラー '%1!'。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EIO

## エラー 16153 ( 廃止予定 )

重大度	N/A
メッセージ	ネットワーク読み取り中にクライアントの接続が切れました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDISC

## エラー 16154 ( 廃止予定 )

重大度	N/A
メッセージ	致命的なエラー：ファイル記述子の最大数を超過しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENETCLOUSE

## エラー 16155 ( 廃止予定 )

重大度	N/A
-----	-----

メッセージ	致命的なエラー：ファイル記述子の最大数を超過しました。
説明	N/A
アクション	N/A
追加情報	記号定数：SRV_ENODEV

## エラー 16156

重大度	10
メッセージ	コマンド引数 (0x%1!) が正しくありません。
説明	いくつかのルーチンは <i>cmd</i> 引数を必要とします。この引数は通常、CS_GET、CS_SET、または CS_CLEAR に設定されています。
アクション	「 <a href="#">第3章 ルーチン</a> 」を参照して、失敗したルーチンに有効な <i>cmd</i> 引数を確認してください。
追加情報	記号定数：SRV_ECMD

## エラー 16157

重大度	10
メッセージ	タイプ引数 (0x%1!) が正しくありません。
説明	無効なデータ型定数が指定されました。
アクション	有効な型定数については、 <a href="#">表 2-37 (229 ページ)</a> を参照してください。
追加情報	記号定数：SRV_ETYPE

## エラー 16158

重大度	10
メッセージ	%1!： CS_DATAFMT におけるカウント (%2!) が正しくありません。
説明	CS_DATAFMT <i>count</i> フィールドに割り当てられた値が無効です。

アクション 有効な *count* フィールド値については、「[CS\\_DATAFMT 構造体](#)」(57 ページ)を参照してください。

追加情報 記号定数: SRV\_EBADCOUNT

## エラー 16159

重大度 10

メッセージ %1!: CS\_DATAFMT におけるステータス (0x%2!) が正しくありません。

説明 CS\_DATAFMT *status* フィールドに割り当てられた値が無効です。

アクション 有効な CS\_DATAFMT ステータスのリストについては、[表 2-9](#) (60 ページ)を参照してください。

追加情報 記号定数: SRV\_EBADSTAT

## エラー 16160

重大度 10

メッセージ %1!: CS\_DATAFMT におけるフォーマット (0x%2!) が正しくありません。

説明 CS\_DATAFMT *format* フィールドに割り当てられた値が無効です。

アクション 有効な CS\_DATAFMT フォーマットのリストについては、[表 2-7](#) (59 ページ)を参照してください。

追加情報 記号定数: SRV\_EBADFORMAT

## エラー 16161 ( 廃止予定 )

重大度 N/A

メッセージ SRV\_MSG に CS\_GET のイベント・ハンドラが存在していません。

説明 N/A

アクション	N/A
追加情報	記号定数: SRV_ENOMSG

## エラー 16162

重大度	10
メッセージ	%1!: %3! の項目番号 (%2!) のバインドが完了しませんでした。
説明	指定されたパラメータがバインドされていません。
アクション	srv_bind() または srv_alt_bind() が指定されたパラメータを正しくバインドしていることを検証してください。
追加情報	記号定数: SRV_ENOBIND

## エラー 16163

重大度	10
メッセージ	データは (%1!) から (%2!) にトランケートされました。
説明	CS_DATAFMT または CS_IODESC の <i>name</i> フィールド、または CS_SERVERMSG の <i>svrname</i> フィールドが CS_MAX_NAME よりも長くなっています。
アクション	CS_MAX_NAME に等しいか、それよりも小さい <i>name</i> または <i>svrname</i> の値を割り当ててください。CS_MAX_NAME の値については、 <i>cstypes.h</i> を参照してください。
追加情報	記号定数: SRV_ETRUNC

## エラー 16164 ( 廃止予定 )

重大度	N/A
メッセージ	NULL プログラム変数でバインドしようとしてしました。
説明	N/A

アクション N/A  
追加情報 記号定数 : SRV\_ENULLADDR

## エラー 16165

重大度 10 または 15  
メッセージ %1!: %3! の %2! が正しくありません。  
説明 指定された **Open Server** ルーチンに対してステータス・エラーが発生しました。  
アクション 指定されたルーチンの有効なステータスについては、「[第3章 ルーチン](#)」を参照してください。  
追加情報 記号定数 : SRV\_ESTATE

## エラー 16166

重大度 10  
メッセージ %1!: 変数、oldvallen(%2!)、newvallen(%3!) の長さが正しくありません。  
説明 `srv_envchange()` が無効な *oldvallen* 値または *newvallen* 値を受け取りました。  
アクション 渡された長さがゼロより大きく、`CS_MAX_CHAR` よりも小さいこと、あるいは長さが `CS_UNUSED` または `CS_NULLTERM` に設定されていることを確認してください。  
追加情報 記号定数 : SRV\_ELEN

## エラー 16167

重大度 10  
メッセージ ロー '[%1!](#)' はすでに記述されています。



説明	<code>srv_alt_header()</code> が重複するローを検出しました。
アクション	ユニークな <i>altid</i> 値を <code>srv_alt_header()</code> に渡してください。
追加情報	記号定数: <code>SRV_EDUPROW</code>

## エラー 16168

重大度	10
メッセージ	%1!: カレント・スレッドの <code>compute</code> ロー '%2!' がありません。
説明	指定されたルーチンは、計算カラムを見つけられません。
アクション	<code>srv_alt_bind()</code> または <code>srv_alt_descfmt()</code> に渡されたカラム番号 (パラメータ <i>item</i> として渡されている) をチェックして、計算カラムに対応していることを確認してください。
追加情報	記号定数: <code>SRV_ENOROW</code>

## エラー 16169

重大度	10
メッセージ	%1!: ID 番号 '%2!' のある集合演算がありません。
説明	<code>srv_alt_descfmt()</code> が無効な集合演算子を受け取りました。
アクション	有効な集合演算子タイプのリストについては、表 3-5 (256 ページ) を参照してください。
追加情報	記号定数: <code>SRV_ENOAGG</code>

## エラー 16170

重大度	10
メッセージ	%1!: テーブル '%2!' はすでに記述されています。
説明	<code>srv_tabname()</code> は、重複するテーブル名を受け取りました。

アクション `srv_tabname()` は、ブラウズ・モードの結果セットに関連付けられたテーブルごとに 1 回だけ呼び出してください。

追加情報 記号定数 : `SRV_EDUPTAB`

## エラー 16171

重大度 10

メッセージ %1!: テーブル '%2!' は記述されていません。

説明 `srv_tabcolname()` は、指定されたテーブルを見つけられません。

アクション `srv_tabname()` を使用してテーブルを定義してください。

追加情報 記号定数 : `SRV_ENOTAB`

## エラー 16172

重大度 10

メッセージ %1!: `compute-id '%2!'` が正しくありません。

説明 `srv_alt_header()` が無効な計算ロー ID を受け取りました。

アクション ゼロより大きい計算ロー ID を渡してください。

追加情報 記号定数 : `SRV_EBADROW`

## エラー 16173 ( 廃止予定 )

重大度 N/A

メッセージ %1!: `compute-by` リストが `order-by` リストに一致していません。

説明 N/A

アクション N/A

追加情報 記号定数 : `SRV_EBADCOMPUTEY`

## エラー 16174

重大度	10
メッセージ	%1!: カレント文に <code>compute-by</code> リストがありません。
説明	<code>srv_alt_header()</code> は、空の <code>bylist</code> 配列を指すポインタを受け取りました。
アクション	<code>bylist</code> 配列内のロー数が、 <code>numbylist</code> で定義した数と同じであることを確認してください。
追加情報	記号定数: <code>SRV_ENOCOMPUTEBY</code>

## エラー 16175

重大度	10
メッセージ	%1!: <code>compute-by</code> カラム数の値 '%2!' が正しくありません。
説明	<code>srv_alt_header()</code> は、指定された <code>bylist</code> カラム数がゼロより小さいために失敗しました。
アクション	<code>numbylist</code> 値が <code>bylistarrayp</code> 配列ポインタ内の要素数と等しいことを確認してください。
追加情報	記号定数: <code>SRV_EBADNUMBYLIST</code>

## エラー 16176

重大度	10
メッセージ	%1!: カレント文に <code>order-by</code> リストがありません。
説明	<code>srv_orderby()</code> は、カラムの空の配列を指すポインタを受け取りました。
アクション	NULL 以外の <code>collistp</code> パラメータを渡してください。
追加情報	記号定数: <code>SRV_ENOORDERBY</code>

## エラー 16177

重大度	10
メッセージ	%1!: order-by カラム数の値 '%2!' が正しくありません。
説明	srv_orderby() に渡されたカラム数がゼロより少ないです。
アクション	ゼロまたはゼロより大きい <i>numcols</i> 値を指定してください。この値は、 <i>collistp</i> 配列内の要素数を示す必要があります。
追加情報	記号定数: SRV_EBADNUMCOLS

## エラー 16178

重大度	10
メッセージ	%1!: テーブル番号 '%2!' が正しくありません。
説明	srv_tabname() に渡されたテーブル ID ( <i>tablename</i> パラメータとして渡されたもの) がゼロより小さいです。
アクション	<i>tablename</i> パラメータはゼロより大きい必要があります。
追加情報	記号定数: SRV_EBADTABNUM

## エラー 16179

重大度	10
メッセージ	%1!: 名前の長さの値 '%2!' が正しくありません。
説明	srv_tabname() が無効な <i>namelen</i> 値を受け取りました。
アクション	<i>namelen</i> を CS_NULLTERM に、またはゼロより大きい値に設定してください。
追加情報	記号定数: SRV_EBADNAMELEN

## エラー 16180

重大度	10
メッセージ	%1!: 制御フォーマットの長さの値 '%2!' が正しくありません。
説明	srv_setcontrol() は、制御日付長さが無効なために失敗しました。
アクション	ctrlen をゼロ以上で、SRV_MAXCHAR より小さい値に設定してください。
追加情報	記号定数: SRV_EBADCTRLLEN

## エラー 16181

重大度	10
メッセージ	%1!: '%2!' 引数が NULL になることはありません。
説明	指定されたルーチンは、NULL 値をもつテーブル名を渡しました。
アクション	srv_tabname() に渡された、または CS_BROWSEDESC name フィールドで定義されたテーブル名が NULL でないことを確認してください。
追加情報	記号定数: SRV_ENULLNAME

## エラー 16182

重大度	10
メッセージ	%1!: カラム '%2!' が srv_tabcolname() によってテーブルとリンクされなければなりません。
説明	指定されたカラムがテーブルにリンクしていません。
アクション	結果ローのカラムでもある結果カラムごとに 1 回、srv_tabcolname() を呼び出してください。
追加情報	記号定数: SRV_EBADTABCOL

## エラー 16183 ( 廃止予定 )

重大度	10
メッセージ	%1!: %3! のオプション (%2!) がサポートされていません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EUNSUPPORTED 今後のために予約済み。

## エラー 16184 ( 廃止予定 )

重大度	N/A
メッセージ	関数 '%2!' にある文字セット %1! がわかりません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOCHARSET

## エラー 16185 ( 廃止予定 )

重大度	N/A
メッセージ	csr ファイル '%1!' ('%2!' 行) におけるフォーマットが正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECSRFILE

## エラー 16186 ( 廃止予定 )

重大度	N/A
-----	-----

メッセージ	文字セット '%1!' の文字属性構造体の構築エラーです。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ECHARATTR

## エラー 16187 ( 廃止予定 )

重大度	N/A
メッセージ	関数 '%2!' には %1! がありましたが、16 進アラビア数字にしなればなりませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENOTHEX

## エラー 16188

重大度	10
メッセージ	関数 '%3!' における引数 %1! %2! が正しくありません。
説明	指定されたルーチンに渡されたパラメータが無効です。
アクション	指定されたルーチンのパラメータに使用可能な値については、「 <a href="#">第 3 章 ルーチン</a> 」を参照してください。
追加情報	記号定数: SRV_EARG

## エラー 16189 ( 廃止予定 )

重大度	N/A
メッセージ	サーバ文字セット '%1!' とクライアント文字セット '%2!' との間の変換はできません。

## エラー 16190 ( 廃止予定 )

---

説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECSNOCVT

## エラー 16190 ( 廃止予定 )

重大度	N/A
メッセージ	'%2!' で文字セット変換 %1! が発生しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECSCVT

## エラー 16191 ( 廃止予定 )

重大度	N/A
メッセージ	関数 '%2!' の戻り (%1!) がわかりません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENORETURN

## エラー 16192

重大度	10
メッセージ	ログイン記録における %1! 文字セット変換エラーです。



説明	このエラーは、クライアント・アプリケーションのログイン・レコード内の文字が Open Server 文字セットに変換できない場合に発生します。たとえば、クライアントの文字セットが utf8 であって Open Server 文字セットが roman8 の場合、utf8 に属する一部の文字は roman8 に変換できないことがあります。
アクション	クライアント・アプリケーションを Ribo ユーティリティを使用して実行してください。変換不能な文字を示す疑問符については、「ログイン・レコード」をレビューしてください。これらの文字をクライアント・アプリケーションから削除するか、Open Server 文字セットを変更してください。
追加情報	記号定数 : SRV_ECSLOGREC

## エラー 16193 ( 廃止予定 )

重大度	N/A
メッセージ	'%1!' は '%2!' ではありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECSMISMATCH

## エラー 16194 ( 廃止予定 )

重大度	N/A
メッセージ	%1! '%2!' の定義ファイルをロードすることはできません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECSLOAD

## エラー 16195

重大度	10
メッセージ	CS_CANBENULL セットのデータ型 (%1!) が正しくありません。
説明	CS_DATAFMT <i>status</i> ビットが CS_CANBENULL に設定されました。
アクション	ブール値である CS_BIT_TYPE を 0 または 1 に設定してください。
追加情報	記号定数 : SRV_EBADNULL

## エラー 16196 ( 廃止予定 )

重大度	N/A
メッセージ	srv_init() の後で構成オプション '%1!' を変更することはできません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECHCONFIG

## エラー 16197 ( 廃止予定 )

重大度	N/A
メッセージ	compute-by リストがある場合、srv_alt_header() の前に %1! が呼び出されなければなりません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ELATEORDER

## エラー 16198

重大度	10
-----	----

メッセージ	項目番号 (%1!) が正しくありません。
説明	srv_alt_bind()、srv_alt_descfmt()、srv_bind()、srv_descfmt()、または srv_text_info() が、 <i>item</i> パラメータに無効な値を受け取りました。
アクション	1 以上であり、カラムまたはパラメータの最大数よりも小さい <i>item</i> 値を指定してください。カラムまたはパラメータの現在の最大数については、Sybase サポート・センタにお問い合わせください。
追加情報	記号定数: SRV_EBADITEM

## エラー 16199 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: 特性 %2! を得るためには SRV_BULK ハンドラになければなりません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENOTINBULK.

## エラー 16200

重大度	10
メッセージ	%1!: 転送タイプ '%2!' が正しくありません。
説明	Open Server は blk_srvinit()、blk_gettext()、blk_colval()、blk_getrow()、srv_get_text()、または srv_text_info() を呼び出しましたが、SRV_T_BULKTYPE 値が SRV_BULKLOAD ではありません。
アクション	SRV_T_BULKTYPE 値ごとに適切なコマンドを呼び出してください。 <ul style="list-style-type: none"> <li>• 1 – SRV_BULKLOAD</li> <li>• 2 – SRV_TEXTLOAD</li> <li>• 3 – SRV_IMAGELOAD</li> <li>• 4 – SRV_UNITEXTLOAD</li> </ul>
	<a href="#">表 2-29 ( 178 ページ )</a> を参照してください。

追加情報 記号定数 : SRV\_EBADIOTYPE

## エラー 16201

重大度 10

メッセージ %1! は単一カラムのローの場合にかぎって呼び出し可能です。

説明 転送されるカラムは、`srv_bind()` または `srv_alt_bind()` を使用してバインドされているため、`srv_send_text()` では送信できません。

アクション `srv_send_text()` を使用して `text` データまたは `image` データを転送する場合は、`srv_text_info()` を呼び出してデータを記述してください。

追加情報 記号定数 : SRV\_EGTONECOL

## エラー 16202

重大度 10

メッセージ %1! : %2! にはなにも残っていません。

説明 `srv_get_text()` に読み込むものが何もないか、`srv_send_text()` に書き込むものが何もありません。

アクション `srv_get_text()` または `srv_send_text()` を呼び出すたびに、`srv_text_info()` で指定された長さを最大として追加される長さを指定してください。`text` カラムまたは `image` カラムに対して `NULL` 値を送信している場合は、ゼロを指定してください。

追加情報 記号定数 : SRV\_ENONELEFT

## エラー 16203

重大度 10

メッセージ %1! : カラムはテキストまたはイメージ・タイプでなくてはなりません。

説明 `srv_send_text()` が無効なカラム型を受け取りました。

アクション	srv_send_text() を呼び出す前に、srv_text_info() で指定したカラムの型が CS_TEXT_TYPE、CS_XML_TYPE、CS_IMAGE_TYPE、または CS_UNITEXT_TYPE であることを確認してください。
追加情報	記号定数 : SRV_EWRONGTYPE

## エラー 16204

重大度	10
メッセージ	%1!: 構造体メンバー '%3!' の値 %2! が正しくありません。
説明	構造体 CS_SERVERMSG または CS_IODESC の指定されたメンバの長さが無効です。
アクション	構造体メンバ CS_SERVERMSG または CS_IODESC に有効な長さを割り当ててください。『Open Client Library/C リファレンス・マニュアル』を参照してください。
追加情報	記号定数 : SRV_EBADNUMVAL

## エラー 16205 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: パラメータ '%2!' はすでに記述されています。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDUPPARAM

## エラー 16206 ( 廃止予定 )

重大度	N/A
メッセージ	RPC パラメータ記述子が完全ではありません。-- %1! %2! が記述されていません。

説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ERPCINC

## エラー 16207

重大度	10
メッセージ	%1!: 読み込まれたローは %2! バイトを超えてはいけません。
説明	blk_getrow() は、指定されたバイト数を超えるローを受け取りました。
アクション	Adaptive Server で dbcc serverlimits を発行し、「テーブル固有のロー・サイズの限度」の項をレビューして、Adaptive Server ローの最大長を決定してください。
追加情報	記号定数 : SRV_EBADROWLEN

## エラー 16208

重大度	10
メッセージ	%1!: compute ロー・カラムはテキストまたはイメージ・タイプにはなりません。
説明	srv_alt_descfmt() は型が text、image、または xml のカラムを受け取りましたが、これは srv_alt_descfmt() では処理できません。
アクション	text、image、または xml を srv_alt_descfmt() に渡さないでください。
追加情報	記号定数 : SRV_EBADALTCOLTYPE

## エラー 16209 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: SRV_PROC の srvp_iotype フィールドが初期化されていません。

説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EIOTYPENOTSET

## エラー 16210

重大度	10
メッセージ	%1!: カラム %2! をバインドしてはなりません。
説明	srv_send_text() は、srv_bind() を使用してバインドされているカラムを送信しようとしてしました。
アクション	srv_bind() によってバインドされた text カラムまたは image カラムを送信するには、srv_xferdata() を使用します。srv_send_text() を介して送信するには、srv_text_info() を使用してカラムをバインドしてください。
追加情報	記号定数 : SRV_EBOUND

## エラー 16211 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: カレント・カラムのテキストまたはイメージの長さがすでにクライアントから読み込まれています。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETXTLENREAD

## エラー 16212

重大度	10
メッセージ	%1!: カレント・カラムのテキストまたはイメージの長さがまだクライアントから読み込まれていません。

説明	<code>srv_get_text()</code> は、 <code>srv_text_info()</code> を呼び出してカラムの長さを読み込んでいないために失敗しました。
アクション	<code>CS_GET</code> を使って <code>srv_text_info()</code> を呼び出して、取得する <code>text</code> データまたは <code>image</code> データの完全な記述を取得してください。
追加情報	記号定数 : <code>SRV_ETXTLENNOTREAD</code>

## エラー 16213

重大度	10
メッセージ	テキストまたはイメージのタイムスタンプが定義されませんでした。
説明	<code>srv_send_text()</code> は、 <code>CS_IODESC</code> 構造体の <code>timestamp</code> フィールドが未定義であることを検出しました。
アクション	<code>timestamp</code> に値を入力してから <code>srv_text_info()</code> を呼び出してください。
追加情報	記号定数 : <code>SRV_ENOTIMESTAMP</code>

## エラー 16214 ( 廃止予定 )

重大度	N/A
メッセージ	使用できる文字セット定義ファイルがありません。 -- サーバの文字セットを '%1!' に設定しています。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_ECSFALLBACK</code>

## エラー 16215 ( 廃止予定 )

重大度	N/A
メッセージ	ログイン・フィールドの '%2!' の範囲外の長さ (%1!) です。



説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ELOGINLEN

## エラー 16216

重大度	10
メッセージ	%1! はテキストまたはイメージ・カラム値を検索することができません。
説明	カラムの値を戻そうとしたときに、blk_colval() が失敗しました。
アクション	blk_colval() は text、image、boundary、sensitivity、unitext、または XML カラムの値を取得できません。blk_gettext() を使用して、これらのカラムを取得してください。
追加情報	記号定数: SRV_EBADCOLTYPE

## エラー 16217

重大度	10
メッセージ	%1!: ロー情報を検索するために blk_getrow() が呼び出されなければなりません。
説明	blk_getrow() の前に blk_colval() または blk_gettext() i が呼び出されます。
アクション	blk_getrow() を使用して CS_BLK_ROW 構造体を入力したあとで、blk_colval() または blk_gettext() を呼び出してください。
追加情報	記号定数: SRV_ENEEDROW

## エラー 16218

重大度	10
メッセージ	%1!: バルク・コピー・オペレーションを初期化するために blk_exec() が呼び出されなければなりません。

説明	<code>blk_getrow()</code> 、 <code>blk_gettext()</code> 、または <code>blk_srvinit()</code> に渡された <code>CS_BLKDESC</code> 構造体が適切に初期化されていません。
アクション	<code>blk_alloc()</code> を呼び出して <code>CS_BLKDESC</code> を初期化し、初期化が正しく実行されたことを確認してください。
追加情報	記号定数 : <code>SRV_ENEEDHANDLE</code>

## エラー 16219

重大度	10
メッセージ	アテンションにより %1! への呼び出しに割り込みが発生しました。
説明	指定された API は、 <code>srv_sleep()</code> を待っている間に割り込まれました。これは、ユーザが <code>Ctrl+C</code> を押したり、アプリケーションから <code>dbcancel()</code> または <code>ct_cancel()</code> が発行されたときのように、クライアント・アプリケーションが突然切断した場合、そして <code>Open Server</code> がスリープしているときのスレッドが読み込み完了からウェイクアップした場合に生じることのある情報メッセージです。
アクション	追加の問題が起こらないかぎり、このエラーを無視してください。
追加情報	記号定数 : <code>SRV_EINTERRUPTED</code>

## エラー 16220

重大度	10
メッセージ	%1!: ステータス引数 (0x%2!) が正しくありません。
説明	<code>srv_senddone()</code> が無効なステータスを受け取りました。
アクション	有効なステータスのリストについては、 <a href="#">表 3-120 (441 ページ)</a> を参照してください。
追加情報	記号定数 : <code>SRV_ESTATUS</code>

## エラー 16221

重大度	10
メッセージ	%1!: SRV_DONE_FINAL で SRV_DONE_FLUSH を設定することはできません。
説明	SRV_DONE_FLUSH を SRV_DONE_MORE と一緒に設定できますが、SRV_DONE_FINAL と一緒に設定することはできません。
アクション	SRV_DONE_FLUSH と SRV_DONE_FINAL が同時に設定されていないことを確認してください。
追加情報	記号定数: SRV_ENORPC

## エラー 16222

重大度	10
メッセージ	%1!: SRV_CONNECT から SRV_DONE_FLUSH を実行することはできません。
説明	アプリケーションが SRV_CONNECT イベント・ハンドラで <code>srv_senddone()</code> を呼び出して、SRV_DONE_FLUSH ビットを設定しました。
アクション	ステータスを SRV_CONNECT の外側の SRV_DONE_FLUSH に設定してください。
追加情報	記号定数: SRV_EFLUSHCON

## エラー 16223

重大度	10
メッセージ	%1!: SRV_DONE_FINAL で SRV_DONE_MORE を設定することはできません。
説明	SRV_DONE_MORE および SRV_DONE_FINAL のステータス・ビットは相互に排他的であり、同時に設定できません。
アクション	<code>srv_senddone()</code> の呼び出し中に、SRV_DONE_MORE と SRV_DONE_FINAL を一度に 1 つずつ設定してください。

追加情報 記号定数 : SRV\_EMORE

## エラー 16224 ( 廃止予定 )

重大度 N/A

メッセージ %1!: SRV\_CONNECT イベントから SRV\_DONE\_MORE を実行することはできません。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_EMORECON

## エラー 16225 ( 廃止予定 )

èdèÀix N/A

メッセージ %1!: コンテキストが正しくありません。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_ECONTEXT

## エラー 16226 ( 廃止予定 )

重大度 N/A

メッセージ %1!: バージョン番号 (%1!) が正しくありません。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_EVERSION

## エラー 16227

重大度	15
メッセージ	spid %2! によって受け取った %1! TDS ストリームがわかりません。
説明	Open Server は無効なトークンを受け取りました。いくつかの TDS トークンは TDS の特定のバージョンのみで有効であり、いくつかは特定のイベント中のみで有効です。
アクション	Ribo を使用してクライアント・アプリケーションと Open Server 間で渡されたトークンを記録し、どのトークンが無効であるか調べてください。また、次の Open Server SRV_S_TRACEFLAG ビット、SRV_TR_TDSHDR、SRV_TR_TDSDATA、および SRV_TR_EVENT を有効にしてください。
追加情報	記号定数 : SRV_EBADTDS

## エラー 16228 ( 廃止予定 )

重大度	N/A
メッセージ	%1! : SRV_OPTION ハンドラでのみ呼び出し可能です。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EINOPTION

## エラー 16229

重大度	10
メッセージ	%1! : %3! における TDS バージョン (%2!) が正しくありません。
説明	srv_thread_props() は、無効な TDS バージョンで SRV_T_TDSVERSION を設定しようとしてしました。
アクション	有効な SRV_T_TDSVERSION 値のリストについては、 <a href="#">表 2-26 ( 167 ページ )</a> を参照してください。
追加情報	記号定数 : SRV_EWRONGTDSVER

## エラー 16230

重大度	15
メッセージ	%1!: 関数 %2! におけるクライアントからのオプション・コマンドがわかりません。
説明	srv_options() は、クライアント・アプリケーションが無効なオプションを設定するよう要求したため、失敗しました。有効なオプションについては、 <a href="#">表 3-78 (367 ページ)</a> を参照してください。
アクション	Ribo を使用して、クライアント・アプリケーションが Open Server に設定するよう要求した無効なオプションを特定してください。
追加情報	記号定数 : SRV_EWRONGOPTCMD

## エラー 16231

重大度	15
メッセージ	%1!: 関数 %2! におけるクライアントからのオプションがわかりません。
説明	Open Server は srv_options() を使用してオプションを設定できませんでした。
アクション	<a href="#">表 3-80 (370 ページ)</a> を参照して、指定されたオプションに関して srv_options() に渡されたパラメータのデータ型と値が正しいことを確認してください。
追加情報	記号定数 : SRV_EWRONGOPT

## エラー 16232

重大度	10
メッセージ	%1!: カレント・トークン・ストリームにはなにもデータがありません。
説明	現在のトークンのネットワーク・ストリームに読み込めるだけ十分なデータが残っているかどうかを判別する内部チェックが失敗しました。これはネットワーク問題であるか、旧 Sybase 製品 (CR #212015 および CR #434920) との非互換性の問題であることが考えられます。

アクション エラーを再現するには、Ribo を使用してトークンに関する情報を取得してください。

追加情報 記号定数: SRV\_ENODATA

## エラー 16233

重大度 10

メッセージ 関数 %2! における引数 %1! は NULL です。

説明 指定された関数は NULL 値を受け取りました。

アクション Open Server コードをチェックして、渡されたパラメータが有効であることを確認してください。

追加情報 記号定数: SRV\_ENULLARG

## エラー 16234

重大度 10

メッセージ ユーザ・イベント・ハンドラで %1! を呼び出すことはできません。

説明 ユーザ定義イベント・ハンドラが結果を送信または受信しようとした。ユーザ定義イベント・ハンドラから結果タイプ (ROWDATA、RPCDATA、DONE\_FINAL など) に関して `srv_senddone()` を送信することは許可されていません。

アクション `srv_senddone()` がユーザ定義イベント・ハンドラから呼び出されていないことを確認してください。

追加情報 記号定数: SRV\_EINUSEREVENT

## エラー 16235

重大度 10

メッセージ 関数 %3! におけるカラム (%2!) のバインド値 (0x%1!) が正しくあ

	りません。
説明	指定されたカラムの NULL インジケータ値が無効です。
アクション	srv_bind() NULL インジケータ変数を CS_GOODDATA または CS_NULLDATA に設定してください。
追加情報	記号定数 : SRV_EBINDARG

## エラー 16236

重大度	10
メッセージ	%1!: ステータス・エラー (0x%2!)
説明	このエラーは、通常は前の Open Server 呼び出しが失敗したために、Open Server API の呼び出し順序が混乱した場合に起こります。
アクション	Open Server エラー・ログをチェックして、前のエラーで API が先に失敗したものがいないか確認してください。
追加情報	記号定数 : SRV_ESTATECHK

## エラー 16237

重大度	10
メッセージ	%1!: srv_version でバージョンが設定されていません。
説明	Open Server バージョンが srv_version() を使用して設定されていません。
アクション	srv_version() の呼び出しが正常に完了していて、有効なバージョンがバージョン・パラメータに対して指定されていることを確認してください。
追加情報	記号定数 : SRV_ENOVERSION

## エラー 16238

重大度	10
-----	----



メッセージ	%1!: イベント (%3!) の結果タイプ (%2!) が正しくありません。
説明	指定されたイベント・ハンドラから Open Server API を呼び出せません。
アクション	エラー・メッセージで指定された API を指定されたイベント・ハンドラから呼び出さないでください。API を呼び出したイベント・ハンドラを判別するには、 <i>ospublic.h</i> の「Open Server イベント・タイプ」の項でイベント番号を検索してください。
追加情報	記号定数: SRV_EBADRESULT

## エラー 16239 ( 廃止予定 )

重大度	N/A
メッセージ	接続サービス・スレッドがスリープできませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ECONFAIL

## エラー 16240

重大度	10 または 20
メッセージ	%2! において Net-Library ルーチン %1! が異常終了しました。
説明	このエラーは、さまざまなネットワーク関連機能が失敗したときに発生します。情報エラーの場合も、致命的エラーの場合も考えられます。このエラーには付随して追加エラーが発生することが多く、解決する必要のある問題が提起されます。
アクション	エラー・ログに記録された基本的な問題を解決してください。その他のエラーが記録されていない場合は、 <b>truss</b> または <b>tusc</b> などのユーティリティを使用して、失敗した TCP API と失敗の理由を判別してください。
追加情報	記号定数: SRV_ENETFUNC

## エラー 16241

重大度	20
メッセージ	スレッド %1! が新しいイベントを得ることができませんでした。
説明	Open Server はメッセージ・キューからイベントを取得しようとしたが、失敗しました。
アクション	メッセージが使用可能になるまで、Open Server をスリープさせておきます。SRV_I_INTERRUPTED が失敗の原因でない場合は、Sybase サポート・センタに連絡してください。
追加情報	記号定数 : SRV_EPROCFAIL

## エラー 16242 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント接続のクローズができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECLOSEFAIL

## エラー 16243 ( 廃止予定 )

重大度	N/A
メッセージ	プロセス %1! の読み取りオペレーションが異常終了しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EREADFAIL

## エラー 16244

重大度	10
メッセージ	プロセス %1! の書き込みオペレーションが異常終了しました。
説明	srv_sendinfo() は、ログイン処理中にエラー・ハンドラから呼び出されたために失敗しました。クライアント・ログインが突然失敗した場合は、クライアント接続に対応している SRV_PROC 構造体が完全に初期化されていない場合があります、srv_sendinfo() が失敗する原因となっています。
アクション	接続ハンドラが完了した後で srv_sendinfo() を呼び出してください。
追加情報	記号定数 : SRV_EWRITEFAIL

## エラー 16245 ( 廃止予定 )

重大度	20
メッセージ	ブロック割り込みができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EBLOCK

## エラー 16246 ( 廃止予定 )

重大度	20
メッセージ	アンブロック割り込みができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EUNBLOCK

## エラー 16247

重大度	10
メッセージ	サイト・ハンドラ <code>spid %1!</code> が新しいチャネルを設定することができませんでした。
説明	<b>Open Server</b> は別の論理チャネルを設定するようというプロトコル要求を受け取りましたが、そのチャネルはすでに終了しているため失敗しました。
アクション	サイト・ハンドラを要求しているサーバのログ・ファイルをチェックして、チャネルが途中で終了した理由を示すエラーがないか確認してください。
追加情報	記号定数: <code>SRV_ECHANNEL</code> .

## エラー 16248

重大度	10
メッセージ	<code>%1!</code> におけるエラー - ファイル記述子 <code>%2!</code> がもはやアクティブではありません。
説明	<code>srv_select()</code> は、下位レベルのオペレーティング・システム API の失敗により、失敗しました。
アクション	オペレーティング・システム・コマンド ( <code>truss</code> など) を使用して、プロセスを追跡し、失敗したオペレーティング・システム API と失敗の理由を確認してください。基本的なエラーを解決する修正アクションを実行してください。また、 <a href="#">CR 277236</a> を参照してください。
追加情報	記号定数: <code>SRV_EPOLL</code>

## エラー 16249 ( 廃止予定 )

重大度	N/A
メッセージ	<code>%1!</code> : 共用ライブラリのグローバル・データにアクセスすることができませんでした。
説明	N/A

アクション	N/A
追加情報	記号定数: SRV_EGLOBAL

## エラー 16250 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: 非同期マネージャを実行することができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_EIOPPOST

## エラー 16251

重大度	10
メッセージ	%1! は関数 %3! に正しくない長さ値 (%2!) をもっています。
説明	srv_props() または srv_thread_props() が <i>buflen</i> パラメータに対して無効な値を受け取りました。
アクション	<i>buflen</i> がゼロより大きく、バッファの長さをしていることを確認してください。
追加情報	記号定数: SRV_EBADLENARG

## エラー 16252

重大度	10
メッセージ	%2! にあるスレッド特性 (%1!) が正しくありません。
説明	srv_thread_props() が <i>property</i> パラメータに対して無効な値を受け取りました。

アクション 有効なプロパティのリストについては、表 3-147 (487 ページ) を参照してください。

追加情報 記号定数 : SRV\_EPROCPROPS

## エラー 16253

重大度 10

メッセージ %1!: サービス・スレッドには特性 %2! は正しくありません。

説明 Open Server アプリケーションはサービス・スレッドで `srv_thread_props()` を呼び出しました。ただし、現在はサービス・スレッドで設定または取得可能なプロパティが存在しません。

アクション サービス・スレッドで `srv_thread_props()` を呼び出さないでください。

追加情報 記号定数 : SRV\_EINSERVTHREAD

## エラー 16254

重大度 10

メッセージ %1!: %2! 特性は SRV\_DISCONNECT ハンドルにかぎって有効です。

説明 `srv_thread_props()` *property* パラメータが SRV\_T\_CLIENTLOGOUT に設定されていますが、`srv_thread_props()` は SRV\_DISCONNECT 以外のイベントから呼び出されています。

アクション `srv_thread_props(SRV_T_CLIENTLOGOUT)` は、SRV\_DISCONNECT イベント・ハンドラのみから呼び出してください。

追加情報 記号定数 : SRV\_EFROMDISCONNECT

## エラー 16255

重大度 10

メッセージ %1!: 特性 %3! のバッファ・サイズが (%2!) よりも小さいです。

説明	<code>srv_dynamic()</code> 、 <code>srv_props()</code> 、または <code>srv_thread_props()</code> に渡された <i>buflen</i> パラメータが指定されたプロパティに対して小さすぎます。
アクション	<code>srv_dynamic()</code> が失敗の原因である場合は、必要なバッファ長さについて表 3-42 (307 ページ) を参照してください。 <code>srv_props()</code> または <code>srv_thread_props()</code> が失敗の原因である場合は、指定されたプロパティのデータ型に対して <i>buflen</i> が適切であることを確認してください。
追加情報	記号定数: <code>SRV_ESMALLBUF</code>

## エラー 16256

重大度	10
メッセージ	%1!: %3! のバッファ・サイズは (%2!) バイトを超えてはいけません。
説明	指定された API に渡された <i>buflen</i> パラメータが最大値よりも大きいです。
アクション	指定された API に渡された <i>buflen</i> がデータ型に適切であることを確認してください。 <code>character</code> データ型の場合、通常、最大値は <code>SRV_MAXCHAR</code> です。
追加情報	記号定数: <code>SRV_ELAREGBUF</code>

## エラー 16257

重大度	10
メッセージ	%1!: 特性 %2! を設定するためには接続ハンドラに存在する必要があります。
説明	いくつかのプロパティ ( <code>SRV_T_PASSTHRU</code> 、 <code>SRV_T_TDSVERSION</code> など) は <code>SRV_CONNECT</code> イベント・ハンドラ内のみで設定可能な場合があります。
アクション	指定されたプロパティを <code>SRV_CONNECT</code> 内で設定してください。
追加情報	記号定数: <code>SRV_EFROMCONNECT</code>

## エラー 16258 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: TDS レベル (%2!) はカレント・レベル (%3!) よりも大きく設定することはできません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EWRONGTDSSET

## エラー 16259 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: 特性 %3! で (%2!) は正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EWRONGLANG

## エラー 16260

重大度	10
メッセージ	%1!: 特性 %2! を設定またはクリアすることはできません。
説明	読み込み専用プロパティを設定しようとしてしました。
アクション	srv_thread_props() プロパティの特性については <a href="#">表 3-147 ( 487 ページ )</a> を、srv_props() の特性については <a href="#">表 3-86 ( 380 ページ )</a> を参照してください。
追加情報	記号定数 : SRV_ENOPROPSET



## エラー 16261

重大度	10
メッセージ	%2! におけるサーバ特性 (%1!) が正しくありません。
説明	srv_props() を介して無効なプロパティを設定する試みが失敗しました。
アクション	有効なプロパティのリストについては、表 3-86 (380 ページ) を参照してください。
追加情報	記号定数: SRV_ESRVPROPS

## エラー 16262

重大度	10
メッセージ	%1!: srv_init() の後でサーバ特性を構成することはできません。
説明	Open Server は、srv_init() が完了した後で srv_props() を使ってプロパティを設定しようとしてしました。
アクション	SRV_S_TRACEFLAG を除き、srv_props() を使って設定可能なプロパティはすべて、srv_init() を呼び出す前に設定してください。
追加情報	記号定数: SRV_EPROPCFG

## エラー 16263

重大度	10
メッセージ	%1!: 特性 %3! の値 (%2!) が正しくありません。
説明	指定された API は非整数バッファを受け取りました。
アクション	srv_thread_props() プロパティの特性については表 3-147 (487 ページ) を、srv_props() プロパティの特性については表 3-86 (380 ページ) を参照してください。
追加情報	記号定数: SRV_EBADINTPROP

## エラー 16264 ( 廃止予定 )

重大度	10
メッセージ	%1!: %2! の機能マスクが正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECAPMASK

## エラー 16265

重大度	10
メッセージ	%1!: このプラットフォームでは特性 %2! はサポートされていません。
説明	srv_props() は SRV_S_PREEMPT を有効化できませんでした。
アクション	Open Server がスレッドで安全なライブラリ ( 名前付き *_r ) を使ってリンクされていることを確認してください。
追加情報	記号定数 : SRV_EPROPNSUPPORT

## エラー 16266

重大度	10
メッセージ	%1!: 特性 %2! をクリアすることはできません。
説明	srv_props() は、指定されたプロパティの CS_CLEAR によるリセットに失敗しました。
アクション	いくつかのプロパティは読み込み専用です。クリア可能なプロパティのリストについては、 <a href="#">表 3-86 ( 380 ページ )</a> を参照してください。
追加情報	記号定数 : SRV_ENOPROPCLEAR

## エラー 16267

重大度	10
メッセージ	%1!: 特性 カレント TDS バージョン (%3!) の %2! が正しくありません。
説明	考えられる原因: <ul style="list-style-type: none"> <li>• <code>srv_negotiate()</code> は、クライアント・アプリケーションが 5.0 よりも前の TDS バージョンを使用したために失敗しました。</li> <li>• <code>srv_thread_props()</code> は、指定したプロパティがクライアント・アプリケーションの TDS レベルでサポートされていないために、そのプロパティの設定に失敗しました。</li> </ul>
アクション	クライアント・アプリケーションをアップグレードしてください。クライアント・アプリケーションが DB-Library™ API を使用している場合は、 <code>dbsetversion(DBVERSION_100)</code> を呼び出してください。
追加情報	記号定数: <code>SRV_ENOPROPTDS</code>

## エラー 16268 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: <code>srv_init</code> の前に特性 %2! を得ることはできません。
説明	N/A
アクション	N/A
追加情報	記号定数: <code>SRV_ENOSERVNAME</code>

## エラー 16269

重大度	15
メッセージ	イベント・ハンドラは最後の <code>DONE</code> を送りませんでした。

## エラー 16270 ( 廃止予定 )

---

説明	Open Server は、 <code>srv_senddone(SRV_DONE_FINAL)</code> が必要時に送信されなかったことを検出しました。通常、これはプログラミング・エラーです。実行パスがイベント・ハンドラ内に存在するため、ハンドラは <code>srv_senddone(SRV_DONE_FINAL)</code> を呼び出さずに終了してしまいます。
アクション	前出のエラーを調べて、どのイベント・ハンドラが <code>SRV_DONE_FINAL</code> を送信しなかったか判別してください。 <code>SRV_TR_EVENT</code> トレース・フラグを有効化して、常に実行しているイベント・ハンドラを確認してください。
追加情報	記号定数 : <code>SRV_ENOFDONE</code>

## エラー 16270 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: <code>SRV_DONE_FLUSH</code> は <code>SRV_DONE_ERROR</code> と同時には設定することはできません。
説明	N/A
アクション	N/A
追加情報	記号定数 : <code>SRV_EERROR</code>

## エラー 16271

重大度	N/A
メッセージ	データベース・コンテキストが '%1!' へ変更されました。
説明	Adaptive Server 5701 の内部使用でデータベース・メッセージが変更されました。
アクション	N/A
追加情報	記号定数 : <code>SRV_EENVDB</code>

## エラー 16272

重大度	N/A
メッセージ	言語設定が '%1!' へ変更されました。
説明	Adaptive Server 5703 の内部使用で言語メッセージが変更されました。
アクション	N/A
追加情報	記号定数 : SRV_EENVLANG

## エラー 16273

重大度	N/A
メッセージ	クライアントの文字セットの設定を '%1!' に変更しました。
説明	Adaptive Server 5704 の内部使用で文字セット・メッセージが変更されました。
アクション	N/A
追加情報	記号定数 : SQLSRV_ENVCHAR

## エラー 16274

重大度	10
メッセージ	%1!: ID %2! または名前 '%3!' を持つカーソルはありません。
説明	指定された API は、指定されたカーソルを見つけられません。
アクション	Open Server は、クライアント・アプリケーションからのカーソル要求の検出と処理に対応します。複数のカーソルが同時に開いている可能性があります。Open Server が SRV_CURSOR イベント・ハンドラで使用しているカーソルが正しいことを確認してください。
追加情報	記号定数 : SRV_ENOCUR

## エラー 16275 (廃止予定)

重大度	N/A
メッセージ	%1!: SRV_CURSOR イベント・ハンドラにありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOCUREVENT

## エラー 16276

重大度	10
メッセージ	%1!: カーソルのパラメータがありません。
説明	クライアント・アプリケーションは、宣言したカーソル・パラメータを送信しませんでした。
アクション	Ribo を実行して、クライアント・アプリケーションが宣言したパラメータを Open Server に送信されたパラメータと比較してください。
追加情報	記号定数 : SRV_ENOCURPARAMS

## エラー 16277 (廃止予定)

重大度	N/A
メッセージ	%1!: カーソルのパラメータ %2! がありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOCURPARAM

## エラー 16278

重大度	10
メッセージ	%1!: カーソルの更新カラムがありません。
説明	srv_descfmt() が SRV_UPCOLDATA の取得に失敗しました。
アクション	Ribo をクライアント・アプリケーションと Open Server の間で実行して、クライアント・アプリケーションが更新用にカラムを宣言し、送信したかどうか確認してください。
追加情報	記号定数: SRV_ENOCURUPCOLS

## エラー 16279

重大度	10
メッセージ	%1!: カーソルの更新カラム %2! がありません。
説明	SRV_CURDESC <i>numupcols</i> フィールドの値が <i>srv_descfmt()</i> に渡された <i>item</i> 値よりも小さいです。
アクション	<i>srv_descfmt()</i> を使用して SRV_UPCOLDATA を取得する場合は、 <i>item</i> が <i>numupcols</i> よりも大きいことを確認してください。
追加情報	記号定数: SRV_ENOCURUPCOL

## エラー 16280

重大度	10
メッセージ	%1!: カーソル '%2!' はすでに宣言されています。
説明	重複する名前のカーソルがすでに宣言されています。
アクション	クライアント・アプリケーションと Open Server の間で Ribo を実行してください。重複する宣言がないか CURDECLARE トークンを検索し、クライアント・アプリケーションのコードを必要に応じて修正してください。
追加情報	記号定数: SRV_EDUPCUR

## エラー 16281

重大度	10
メッセージ	%1!: カーソル記述子が正しくありません。
説明	srv_cursor_props() は、現在のカーソル・コマンド ( <i>curcmd</i> ) が無効なコマンド・オプション ( <i>cmdoptions</i> ) に関連付けられているために失敗しました。
アクション	指定された <i>curcmd</i> に対して有効な <i>cmdoptions</i> のリストについては、 <a href="#">表 2-14 (75 ページ)</a> を参照してください。
追加情報	記号定数 : SRV_EBADCUR

## エラー 16282

重大度	10
メッセージ	%1!: カーソルのロー記述子がありません。
説明	クライアント・アプリケーションと Open Server 間の接続は、Open Server がローを記述した TDS トークンを受け取る前に失敗した可能性があります。
アクション	エラーを再現できる場合は、Open Server とクライアント・アプリケーション間で Ribo を実行して、クライアント・アプリケーションが ROWFMT トークンを送信したか確認してください。
追加情報	記号定数 : SRV_ENOCURROWDESC

## エラー 16283

重大度	10
メッセージ	%1!: カーソルのロー・フェッチ・カウント %2! が限界を超えました。
説明	srv_xferdata() は、クライアント・アプリケーションが CS_CURSOR_ROWS プロパティに対して設定したロー数よりも多くのローを送信しようとしていました。クライアント・アプリケーションが CS_CURSOR_ROWS プロパティを設定している場合、Open Server はこの要求に準拠する必要があります。



アクション	このタイプの要求処理に適切な呼び出し順については、表 2-15 (78 ページ) を参照してください。クライアント・アプリケーションがフェッチ・カウントを設定している場合の <code>srv_xferdata()</code> の使用方法については、 <code>ctos</code> をレビューしてください。
追加情報	記号定数: <code>SRV_EROWCNT</code>

## エラー 16284

重大度	10
メッセージ	%1!: DONE を送る前に <code>srv_cursor_props()</code> はカーソル情報を送らなければならない。
説明	Open Server は、必要な <code>srv_cursor_props(CS_SET)</code> 呼び出しが完了する前に <code>srv_senddone()</code> を呼び出しました。
アクション	次のことを確認します。 <ul style="list-style-type: none"><li>• Open Server は、<code>srv_senddone()</code> を呼び出す前に、表 2-15 (78 ページ) に示す必須の <code>srv_cursor_props()</code> 応答をすべて完了しています。</li><li>• CR #310107 の影響を受けないバージョンの Open Server を使用しています。</li></ul>
追加情報	記号定数: <code>SRV_ENOCURSENT</code>

## エラー 16285

重大度	10
メッセージ	%1!: <code>srv_cursor_props()</code> によってカーソル情報はすでに送られました。
説明	<code>srv_cursor_props()</code> を複数回呼び出すと、このエラーが発生することがあります。
アクション	不要な <code>srv_cursor_props()</code> 呼び出しを削除してください。
追加情報	記号定数: <code>SRV_ECURSENT</code>

## エラー 16286

重大度	10
メッセージ	%1!: カーソル情報の送信エラーです。
説明	srv_cursor_props() は、クライアント・アプリケーションの CS_CURSOR_INFO 要求に応答して TDS トークンをサーバに送信したときに失敗しました。通常、このエラーは一時的なネットワーク・エラーによって生じます。
アクション	ネットワーク切断を示唆するようなその他のエラーがないか調べてください。エラーを再現できる場合は、 <b>Open Server</b> と送信先サーバ間で <b>Ribo</b> を実行して詳細情報を取得してください。
追加情報	記号定数: SRV_ENOSENDCUR

## エラー 16287 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: カレント・カーソル・ローのキー・カラムが記述されませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENOKEY

## エラー 16288 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: キー・カラム %2! がありません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENOKEYCOL

## エラー 16289

重大度	10 または 20
メッセージ	%1!: 動的 SQL 文 '%2!' がありません。
説明	srv_dynamic() が指定された文を見つけられない場合、重大度は 20 です。srv_cursor_props() が指定された文を見つけられない場合、重大度は 10 です。
アクション	srv_dynamic() に指定された SRV_DYN_ID が有効であることを確認してください。srv_cursor_props() を使用している場合、cursor 文のテキストは実際には作成された文の名前になります。その名前が宣言時の名前と同じであることを確認してください。
追加情報	記号定数: SRV_ENODYN

## エラー 16290 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: SRV_DYNAMIC イベント・ハンドラにありません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENODYNEVENT

## エラー 16291

重大度	10
メッセージ	%1!: 動的 SQL パラメータがありません。
説明	リモート・サーバから動的入力パラメータまたは出力パラメータが Open Server に返されましたが、Open Server はこれらのパラメータをクライアント・アプリケーションに返すときにパラメータの記述に失敗しました。
アクション	常に srv_descfmt(SRV_DYNDATA) を呼び出してパラメータを記述してください。

追加情報 記号定数 : SRV\_ENODYNPARAMS

## エラー 16292 ( 廃止予定 )

重大度 N/A

メッセージ %1!: 動的 SQL パラメータ %2! がありません。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_ENODYNPARAM

## エラー 16293

重大度 10

メッセージ %1!: 動的 SQL 文 '%2!' はすでに準備されています。

説明 Open Server は、重複する文 ID が含まれている DYNAMIC トークンをクライアント・アプリケーションから受け取りました。クライアント・ライブラリでステータス・チェックを実行すると、通常、このエラーは発生しません。

アクション 問題を再現できる場合は、クライアント・アプリケーションと Open Server 間で Ribo を実行して、クライアント・アプリケーションから送信される DYNAMIC トークン関連の情報を取得してください。

追加情報 記号定数 : SRV\_EDUPDYN

## エラー 16294 ( 廃止予定 )

重大度 N/A

メッセージ %1!: 動的 SQL パラメータ・リストが不完全です。%2! パラメータが記述されていません。

説明 N/A

アクション	N/A
追加情報	記号定数: SRV_EDYNINC

## エラー 16295

重大度	15
メッセージ	%1!: 動的 SQL 情報の送信エラーです。
説明	Open Server は、おそらくネットワーク・エラーまたは文字セット変換エラーのため、 <b>DYNAMIC</b> トークンをクライアント・アプリケーションに送信できませんでした。
アクション	クライアントのエラー・ログで、どのタイプの失敗が発生したかを示唆する追加エラーを調べてください。エラーを再現できる場合は、クライアント・アプリケーションと Open Server 間で <b>Ribo</b> を実行すると、エラーの修正方法 (文字セットの変更など) を判別するのに役立ちます。
追加情報	記号定数: SRV_ENOSENDDYN

## エラー 16296

重大度	10
メッセージ	%1!: %2! を送る前に <code>srv_dynamic()</code> によって動的 SQL 文の情報が送られなければなりません。
説明	Open Server は、 <code>srv_xferdata()</code> 、 <code>srv_senddone()</code> 、または <code>srv_descfmt()</code> の呼び出しを試みる前に、必須の <code>srv_dynamic()</code> 呼び出しの 1 つを正常に呼び出しませんでした。
アクション	<code>srv_dynamic()</code> 呼び出しの適切な順序については、 <a href="#">表 2-19 (92 ページ)</a> を参照してください。
追加情報	記号定数: SRV_ENODYNSENT

## エラー 16297

重大度	10
メッセージ	%1!: srv_dynamic() によって動的 SQL 文の情報はすでに送られました。
説明	Open Server は、同じ情報に対して srv_dynamic() を 2 回以上呼び出しています。
アクション	Open Server の srv_dynamic() の使用状況をチェックして、同じ情報が 2 回以上送信されることのないようにしてください。srv_dynamic() 呼び出しの適切な順序については、表 2-19 (92 ページ) を参照してください。
追加情報	記号定数 : SRV_EDYNSENT

## エラー 16298

重大度	10
メッセージ	%1!: この %2! の呼び出しが正しくありません。
説明	指定された API は、指定されたタイプのコマンドに対して不適切に呼び出されています。
アクション	指定された API の適切な使用については、「第 3 章 ルーチン」をレビューしてください。たとえば、エラーがカーソル・コマンド実行時の srv_langlen() の呼び出しに関連している場合、srv_langlen() はカーソルの宣言文または更新文での言語文字列の処理のみに使用できるとマニュアルに記載されています。
追加情報	記号定数 : SRV_EBADCTXT

## エラー 16299 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: '%3!' への応答 '%2!' が正しくありません。
説明	N/A
アクション	N/A

追加情報 記号定数 : SRV\_EBADRESP

## エラー 16300

重大度 15

メッセージ クライアントから受け取ったトークン 0x%1! がわかりません。

説明 Open Server は、カーソル関連の TDS トークンを読み込もうとしているときに、無効または不明のトークンをクライアント・アプリケーションから受け取りました。

アクション エラーを再現できる場合は、クライアント・アプリケーションと Open Server 間で Ribo を実行して、エラーの原因となるトークンがどれであるか確認してください。エラーが再現できない場合は、ネットワークまたはクライアント・アプリケーションのメモリ破壊が原因と考えられます。

追加情報 記号定数 : SRV\_EBADTOKEN

## エラー 16301

重大度 10

メッセージ %1! : SRV\_CONNECT イベント・ハンドラにありません。

説明 srv\_setloginfo() が SRV\_CONNECT 接続イベント・ハンドラの外側で呼び出されました。

アクション SRV\_CONNECT イベント・ハンドラのみから srv\_setloginfo() を呼び出してください。

追加情報 記号定数 : SRV\_ENOCONN

## エラー 16302

重大度 10

メッセージ %1! : カーソル ID %2! はすでに割り当てられています。

説明	Open Server は、クライアント・アプリケーションからの <code>srv_cursor_props()</code> によるカーソル宣言の要求に応答している間に、重複するカーソル ID を検出しました。カーソル ID は、接続ハンドルごとにユニークである必要があります。
アクション	クライアント・アプリケーションと Open Server 間で Ribo を実行して、クライアント・アプリケーションが各カーソルをどのように宣言しているか確認してください。
追加情報	記号定数 : <code>SRV_EDUPCURID</code>

## エラー 16303

重大度	10
メッセージ	テキストまたはイメージ・カラム %1! が完全には送られませんでした。
説明	<code>srv_send_text()</code> 、 <code>srv_text_info()</code> 、または <code>srv_senddone()</code> によって、指定されたカラムの <code>text</code> データまたは <code>image</code> データが完全に送信されなかったことが検出されました。 <code>text</code> データと <code>image</code> データは、 <code>srv_bind()</code> の後に <code>srv_xferdata()</code> 、 <code>srv_send_data()</code> 、または <code>srv_send_text()</code> を使用してクライアント・アプリケーションに転送されています。
アクション	Open Server によるこれらのルーチンの使用状況をレビューして、指定されたカラムのデータがすべて読み込まれ、送信されていることを確認してください。
追加情報	記号定数 : <code>SRV_ETEXTLEFT</code>

## エラー 16304

重大度	10
メッセージ	%1!: <code>compute-id %2!</code> が限界を超えました。-- 1 と %3! の間になければなりません。
説明	<code>srv_alt_header()</code> は、有効範囲内に入っていない <code>altid</code> 値を受け取りました。
アクション	<code>altid</code> を正しく初期化して、指定範囲内にあることを確認してください。
追加情報	記号定数 : <code>SRV_EBADALTID</code>



## エラー 16305

重大度	10
メッセージ	%1!: カラム %2! に NULL データが送られました。-- テキスト・ポインタの長さが 0 でした。
説明	CS_IODESC の <i>textptrlen</i> フィールドがゼロに設定されているため、 <i>srv_send_text()</i> はサーバに NULL カラムを送信します。ただし、 <i>srv_send_text()</i> に渡される <i>buflen</i> 値はゼロではありません。つまり、送信対象のデータが存在することを示しています。
アクション	<i>srv_text_info()</i> に記述されたデータとデータ長を、 <i>srv_send_text()</i> に渡された <i>buffer</i> パラメータおよび <i>buflen</i> パラメータと比較してください。これらのパラメータがすべて一致することを確認してください。
追加情報	記号定数 : SRV_ENOTXTSENT

## エラー 16306

重大度	10
メッセージ	%1!: テキストまたはイメージ・ポインタが定義されていません。
説明	<i>srv_send_text()</i> は、 <i>srv_text_info()</i> と同じ値を使用する必要のある CS_IODESC の <i>textptrlen</i> フィールドがゼロより小さいことを検出しました。
アクション	<i>srv_text_info()</i> に渡される CS_IODESC 構造体の <i>textptrlen</i> の値を訂正してください。
追加情報	記号定数 : SRV_ENOTEXTPTR

## エラー 16307 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: カレント RPC にはパラメータがありません。: %2!
説明	N/A
アクション	N/A

追加情報 記号定数 : SRV\_ENOPARAMRPC

## エラー 16308 ( 廃止予定 )

重大度 N/A

メッセージ %2! で機能 (%1!) が正しくありません。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_ECAPABILITY

## エラー 16309

重大度 10

メッセージ %2! で機能タイプ (%1!) が正しくありません。

説明 Open Server は、TDS\_CAPABILITY トークンの読み込み中に無効な TDS 機能タイプを検出しました。

アクション

- クライアント・アプリケーションと Open Server の間で Ribo を実行してください。
- ログイン・レコードを調べて、有効な Sybase クライアントがサーバに接続しようとしていることを確認してください。
- CAPABILITY トークンの *Type* フィールドを調べて、CAP\_REQUEST または CAP\_RESPONSE のいずれかに設定されていることを確認してください。いずれにも設定されていない場合は、パケット破壊を生じたネットワーク・エラーがないか確認してください。

追加情報 記号定数 : SRV\_ECAPTYPE

## エラー 16310

重大度	15
メッセージ	%1!: spid %3! の %2! 中の致命的エラーです。
説明	Open Server が機能トークン、オプション・コマンド、またはパラメータ・フォーマット・トークンなどのデータ・ストリームを読み込んでいるときに、ストリームが途中で終了しました。このエラーは通常、Open Server がクライアントから読み込んでいる間にクライアント・アプリケーションが突然切断したときに発生します。このエラーは、ネットワーク破壊によって生じることがあります。
アクション	クライアント・アプリケーションと Open Server のログインおよびコマンド・タイムアウトを増やしてください。
追加情報	記号定数: SRV_EBADIO

## エラー 16311 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: msgid の値 (%2!) が正しくありません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_EBADMSGID

## エラー 16312 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: カレント・メッセージのパラメータがありません。
説明	N/A
アクション	N/A
追加情報	記号定数: SRV_ENOPARAMMSG

## エラー 16313 (廃止予定)

重大度	N/A
メッセージ	%1!: カレント %3! にパラメータ番号 %2! がありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOPARAMNUM

## エラー 16314 (廃止予定)

重大度	N/A
メッセージ	%1!: パラメータ記述が完全ではありません。%2! %3! が記述されていません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EPARAMINC

## エラー 16315

重大度	15
メッセージ	%1!: プロトコル・エラーが発生しました。: %2!
説明	通常、このエラーは非 Sybase 製品が Open Server にパケットを送信したときに発生します。telnet hostname port または他のポートの監視ソフトウェアによってこのエラーが立ち上げられます。Sybase ベースの製品のみを Open Server のリスニング・ポートに接続してください。
アクション	非 Sybase 製品がサーバに接続しているためにこのエラーが発生した場合は、Open Server のコードを入力して、このエラーを無視するか抑制してください。エラーがさらに重大な問題につながる場合は、ネットワーク・スニファを使用して Open Server に接続しようとしているソフトウェアを特定し、Sybase サポート・センタに連絡してください。
追加情報	記号定数 : SRV_EPROTERR

## エラー 16316 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: カレント・エラー・メッセージのパラメータがありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOPARAMEED

## エラー 16317

重大度	10
メッセージ	%1!: 動的 SQL データは CS_DESCIN ( 入力パラメータ・フォーマット ) または CS_DESCOUT ( 出力カラム・フォーマット ) として定義されなければなりません。
説明	srv_xferdata() または srv_descfmt() は、CS_DATAFMT の <i>status</i> フィールドが CS_DESCIN または CS_DESCOUT ( 入力パラメータまたは出力パラメータを含む動的 SQL 文に必須 ) に設定されていないことを検出しました。
アクション	有効な動的 SQL 要求および応答のリストについては、 <a href="#">表 2-19 (92 ページ)</a> を参照してください。
追加情報	記号定数 : SRV_ENODYNDESC

## エラー 16318 ( 廃止予定 )

重大度	N/A
メッセージ	%1!: blk_gettext() によって読み込まれるテキストまたはイメージ・カラムがまだ残っています。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETXTLEFT

## エラー 16319

重大度	10
メッセージ	%1!: テーブル・カラム情報の送信が異常終了しました。
説明	blk_srvinit() は、テーブル・カラム情報を送信できませんでした。クライアント・アプリケーションからの insert 文に with noddescribe 句が含まれていないかぎり、Open Server は blk_srvinit() を使用して、クライアント・アプリケーションに対して現在のサーバ・テーブル・カラムの記述を CS_BLKDESC 構造体に含める必要があります。
アクション	blk_srvinit() の失敗の基本的な原因を示す追加エラー ( ネットワークの切断など ) がないか調べてください。
追加情報	記号定数 : SRV_EBLKINIT

## エラー 16320

重大度	10
メッセージ	ロケール %1! のローカライゼーション情報をロードすることができませんでした。
説明	指定された文字セット、言語、またはロケール名が見つかりません。
アクション	指定されたローカライゼーション情報に関して \$SYBASE/locales を調べて、ファイルの読み込みパーミッションが有効になっていることを確認してください。「 <a href="#">国際化サポート</a> 」(111 ページ) をレビューしてください。
追加情報	記号定数 : SRV_ENONATLOCALE

## エラー 16321

重大度	10
メッセージ	CS_LOCALE が正しくないか、または、Open Server CS_CONTEXT を使用して割り当てられていません。
説明	srv_thread_props() は、CS_LOCALE ポインタが無効なため、SRV_T_LOCALE 値を設定または取得できませんでした。

アクション `cs_loc_alloc()` を使用して、`CS_LOCALE` 構造体の割り当てと初期化を適切に実行してください。

追加情報 記号定数: `SRV_EBADLOCALE`

## エラー 16322

重大度 10

メッセージ スレッド %1! のローカライゼーション情報を割り当てることができませんでした。

説明 `cs_locale()` または `cs_loc_alloc()` の呼び出しが失敗しました。

アクション `truss` などのトレース・ユーティリティを使用して、エラーの原因を特定してください。

- メモリ不足のため、ロケール情報をロードできません。
- ローカライゼーション・ファイルが見つかりません。
- `cs_locale()` の *action* パラメータは `CS_SET` に設定されていますが、*\*buffer* ロケール名が Sybase の *locales* ファイルに見つかりません。
- `cs_locale()` の *action* パラメータは `CS_GET` に設定されていますが、*buflen* は *\*buffer* データ領域が小さすぎることを示しています。

追加情報 記号定数: `SRV_ENOLOCALE`

## エラー 16323

重大度 10

メッセージ 文字セット %1! または言語 %2! のローカライゼーション情報をロードすることができませんでした。

説明 `sp_serverinfo` または `cs_locale()` は、指定された文字セットまたは言語を見つけれません。

アクション	<code>truss</code> などのトレース・ユーティリティを使用して、 <code>Open Server</code> が文字セット情報を検索している場所を調べ、ファイル・パーミッションが正しいかどうかを確認してください。 <code>locales</code> ファイルが存在していて読み込み可能な場合には、『 <code>Open Client / Server</code> 開発者用国際化ガイド』および「 <a href="#">国際化サポート</a> 」(111 ページ)をレビューして要求の有効性を確認してください。
追加情報	記号定数 : <code>SRV_ENOCHARLOCALE</code>

## エラー 16324

重大度	10
メッセージ	%1!: カラム番号 %2! のデータが <code>NULL</code> になることはありません。
説明	指定された API は、 <code>NULL</code> に設定できないカラムに <code>NULL</code> データが渡されようとしていることを検出しました。
アクション	リモート・カラム定義が <code>NULL</code> を受け入れることを確認してください。 <code>NULL</code> が受け入れられる場合は、 <code>CS_DATAFMT</code> の <code>status</code> フィールドの <code>CS_CANBENULL</code> ビットが設定されていることを確認してください。指定された API が <code>indicator</code> パラメータに <code>NULL</code> 値を受け入れる場合は、 <code>NULL</code> データが受け渡し中であることを示すように正しく設定されていることを確認してください。
追加情報	記号定数 : <code>SRV_EWRONGNULL</code>

## エラー 16325

重大度	10
メッセージ	<code>srv_init</code> が呼び出されていません。
説明	<code>srv_init()</code> が完了する前に <code>Open Server API</code> を呼び出そうとしました。
アクション	<code>srv_init()</code> を使用して <code>Open Server</code> を初期化した後で、 <code>srv_props()</code> と <code>srv_version()</code> を除き、その他の API を呼び出してください。
追加情報	記号定数 : <code>SRV_ENOINIT</code>



## エラー 16326

重大度	15
メッセージ	%2! のときに %1! へ渡された引数の長さが正しくありません。
説明	srv_options() は、クライアント・アプリケーションからオプション・コマンドを読み込んでいる間に、無効なパラメータ長を検出しました。
アクション	クライアント・アプリケーションと Open Server の間で Ribo を実行してください。OPTIONCMD トークンに関して「引数長さ」をレビューし、必要に応じてクライアント・アプリケーションを修正してください。
追加情報	記号定数: SRV_EWRONGARGLEN

## エラー 16327

重大度	10
メッセージ	%1!: クライアントによって送られたリモート情報はありません。
説明	srv_negotiate() が SRV_NEG_REMPWD を取得しようとしたましたが、クライアント・アプリケーションは SRV_NEG_LOCPWD を送信していません。
アクション	SRV_NEG_ENCRYPT チャレンジに対する SRV_NEG_LOCPWD の応答が常に存在することを確認してください。
追加情報	記号定数: SRV_ENONEGPWD

## エラー 16328

重大度	10
メッセージ	%1!: クライアント応答の待機が異常終了しました。
説明	srv_negotiate() は、クライアント・アプリケーションから応答を待っている間に失敗しました。
アクション	Open Server がクライアント・アプリケーションの応答を受信できないようなネットワークの割り込みがあるか確認してください。エラーを再現できる場合は、クライアント・アプリケーションと Open Server 間で Ribo を実行して、パスワード暗号化のチャレンジ中のトークン交換を確認してください。

追加情報 記号定数 : SRV\_EREADFAILED

## エラー 16329

重大度 10

メッセージ %1!: %3! への応答で受け取られた応答タイプ %2! が正しくありません。

説明 `srv_negotiate()` は、指定されたタイプのチャレンジに対して不適切な応答タイプを受け取りました。

アクション エラーを再現できる場合は、クライアント・アプリケーションと Open Server 間で `Ribo` を実行して、チャレンジ中に交換されるトークンを確認してください。正しい応答タイプについては、『Open Client Library/C リファレンス・マニュアル』> Client-Library トピックス > セキュリティ機能 > Adaptive Server Enterprise セキュリティ機能を参照してください。

追加情報 記号定数 : SRV\_EWRONGMSG

## エラー 16330

重大度 15

メッセージ `spid %1!` のパススルーが完全ではありません。

説明 Open Server は、パススルー・モードで実行していた指定の `spid` の後でクリーンアップを行っている間に、トークンが完全に読み込まれていないことを検出しました。

アクション `infop` 値が `SRV_I_PASSTHRU_EOM` になるまで `srv_recvpassthru()` または `srv_sendpassthru()` を呼び出します。この値は、すべてのトークンがクライアント・アプリケーションから読み込まれたことを示します。

追加情報 記号定数 : SRV\_EINCPASSTHRU

## エラー 16331

重大度	10
メッセージ	<code>srv__seterr()</code> に対する再帰呼び出しです。
説明	Open Server エラー・ハンドラは、2 番目のエラー発生時にエラー・ハンドラを再度 (再帰的に) トリガするというエラーを報告しています。たとえば、エラー・ハンドラがクライアント・アプリケーションにエラーを返そうとしていても、クライアントがネットワークから切断されたため、エラー・ハンドラが再度トリガーされて、エラーをクライアント・アプリケーションに送信できないと報告します。
アクション	最初のエラーを起こした状況を修正してください。
追加情報	記号定数: <code>SRV_ERECURSIVE</code>

## エラー 16332

重大度	10
メッセージ	<code>%1! : %2!</code> 結果タイプが保留されているか、または、サポートされていません。
説明	<code>srv_descfmt()</code> または <code>srv_msg()</code> が、指定された結果タイプが保留されていると合図しています。
アクション	クライアント・アプリケーションと Open Server 間で <code>Ribo</code> を実行して <code>CAPABILITY</code> トークンをレビューし、クライアント・アプリケーションと Open Server がサポートを同意している機能を確認してください。
追加情報	記号定数: <code>SRV_ENOESCAP</code>

## エラー 16333

重大度	10
メッセージ	<code>%1! : SRV_START</code> ハンドラで呼び出すことはできません。
説明	指定されたルーチンが <code>SRV_START</code> イベント・ハンドラ内で呼び出せません。

アクション 「第3章 ルーチン」を参照して、指定されたルーチンが使用可能な場所を確認してください。

追加情報 記号定数 : SRV\_EINSTART

## エラー 16334

重大度 10

メッセージ SRV\_START イベントにより `srv_run` がアボートされました。

説明 `srv_run()` は、SRV\_START イベント・ハンドラが失敗したために突然終了しました。

アクション SRV\_START イベント・ハンドラの失敗の基本的な原因を示唆する追加のエラー・メッセージを確認してください。これらの失敗状況を修正してください。

追加情報 記号定数 : SRV\_ESTARTEVENT

## エラー 16335

重大度 10

メッセージ %1!: 項目番号 %2! のデータ長が 0 のときに、NULL インジケータは設定されません。

説明 指定されたルーチンは、指定されたアイテムのデータ長がゼロのときに NULL インジケータ・パラメータが適切に設定されていないことを検出しました。

アクション 指定されたルーチンによる NULL インジケータ・パラメータの使用法については、「第3章 ルーチン」を参照してください。

追加情報 記号定数 : SRV\_ENONULLIND

## エラー 16336

重大度 10

メッセージ	%1!: CS_DATAFMT において総桁数の値 (%2!) が正しくありません。
説明	指定された API は、CS_NUMERIC_TYPE 変数または CS_DECIMAL_TYPE 変数に対して無効な CS_DATAFMT <i>precision</i> 値を検出しました。
アクション	CS_MIN_PREC から CS_MAX_PREC まで、有効な総桁数値を定義する <i>cstypes.h</i> を確認してください。デフォルトの総桁数は CS_DEF_PREC です。
追加情報	記号定数: SRV_EBADPREC

## エラー 16337

重大度	10
メッセージ	%1!: CS_DATAFMT において小数点以下桁数の値 (%2!) が正しくありません。
説明	指定された API は、CS_NUMERIC_TYPE 変数または CS_DECIMAL_TYPET 変数に対して無効な CS_DATAFMT <i>scale</i> 値を検出しました。 <i>scale</i> は <i>precision</i> 以下である必要があります。
アクション	CS_MIN_SCALE から CS_MAX_SCALE まで、有効な小数点以下桁数値を定義する <i>cstypes.h</i> を確認してください。デフォルトの小数点以下桁数は CS_DEF_SCALE です。
追加情報	記号定数: SRV_EBADSCALE

## エラー 16338

重大度	20
メッセージ	srv_run により致命的エラーが見つかりました。
説明	srv_run() での致命的エラーのため、Open Server は起動できません。
アクション	この障害の基本的な原因を示唆する追加のエラー・メッセージ(ファイルの限度やメモリ不足など)を確認してください。
追加情報	記号定数: SRV_ERUN

## エラー 16339

重大度	20
メッセージ	%1! でコルーチン制御を戻すことができませんでした。
説明	これは、プリエンプティブ Open Server にとって致命的なエラーです (SRV_S_PREEMPT=CS_TRUE)。
アクション	このエラーは内部の Open Server バグを示しています。Sybase サポート・センタにレポートしてください。
追加情報	記号定数 : SRV_ETAKECOROUTINE

## エラー 16340

重大度	20
メッセージ	%1! でコルーチン制御を戻すことができませんでした。
説明	これは、プリエンプティブ Open Server にとって致命的なエラーです (SRV_S_PREEMPT=CS_TRUE)。
アクション	このエラーは内部の Open Server バグを示しています。Sybase サポート・センタにレポートしてください。
追加情報	記号定数 : SRV_ERELEASECOROUTINE

## エラー 16341 ( 廃止予定 )

重大度	N/A
メッセージ	%1! でシステム mutex を取ることができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETAKEMUTEX

## エラー 16342 ( 廃止予定 )

重大度	N/A
メッセージ	%1! でシステム mutex を解除することができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ERELEASEMUTEX

## エラー 16343 ( 廃止予定 )

重大度	N/A
メッセージ	spid %1! への応答にエラーがありました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EMMSGSENDREPLY

## エラー 16344 ( 廃止予定 )

重大度	N/A
メッセージ	不適当な読み取りバッファが解放されました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EFREEBUF

## エラー 16345 ( 廃止予定 )

重大度	N/A
-----	-----

## エラー 16346 ( 廃止予定 )

---

メッセージ	現在利用可能なネットワーク読み取りバッファはありません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ENOREADBUFFERS

## エラー 16346 ( 廃止予定 )

重大度	N/A
メッセージ	dead 状態かまたは接続を解除されたクライアントのメッセージを受け取りました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDEADCLIENT

## エラー 16347 ( 廃止予定 )

重大度	N/A
メッセージ	spid %1! のメッセージが省略されました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EMMSGSKIPPED

## エラー 16348 ( 廃止予定 )

重大度	N/A
メッセージ	procid %1! へログイン受け取りメッセージを送ることができませんでした。



説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECLIENTACCEPT

## エラー 16349 ( 廃止予定 )

重大度	N/A
メッセージ	spid %1! は完全に終了しているか、または、接続が解除されています。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ECLIENTDISCONNECT

## エラー 16350 ( 廃止予定 )

重大度	N/A
メッセージ	ダイレクト・キュー %1! の作成エラーです。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EQUEUECREATE

## エラー 16351 ( 廃止予定 )

重大度	N/A
メッセージ	s\$control() とのポートの設定エラーです。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ESETTINGFASTQ

## エラー 16352 ( 廃止予定 )

重大度	N/A
メッセージ	TCP/IP ネットワーク制御構造体のメモリ割り当てができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETCPALLFAIL

## エラー 16353 ( 廃止予定 )

重大度	N/A
メッセージ	TCP/IP デバイス %1! の割り当てができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETCPNETASSIGN

## エラー 16354 ( 廃止予定 )

重大度	N/A
メッセージ	TCP/IP ソケット・オプション SO_KEEPALIVE の設定ができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETCPSETKEEP

## エラー 16355 ( 廃止予定 )

重大度	N/A
-----	-----

メッセージ	TCP/IP ソケット・オプション SO_REUSEADDR の設定ができません。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETCPSETREUSE

## エラー 16356 ( 廃止予定 )

重大度	N/A
メッセージ	TCP/IP 受け入れ待ちエラーです。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETCPACCEPTW

## エラー 16357 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント接続の新しいチャネルの割り当てができませんでした。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_ETCPASSIGN

## エラー 16358 ( 廃止予定 )

重大度	N/A
メッセージ	クライアント接続で利用可能なリンク・エントリーはありません。
説明	N/A
アクション	N/A

追加情報 記号定数 : SRV\_ETCPNOLINK

## エラー 16359 ( 廃止予定 )

重大度 N/A

メッセージ ソケットでアテンション AST の設定ができませんでした。

説明 N/A

アクション N/A

追加情報 記号定数 : SRV\_ETCPATTNAST

## エラー 16360

重大度 20

メッセージ %1! で WaitForSingleObject() が異常終了しました。

説明 WaitForSingleObject() または comn\_waitfor\_event() が指定された API で失敗しました。指定されたミューテックス・オブジェクトを所有していたスレッドが、ミューテックスを解放する前に終了しました。

アクション 所有スレッドがオブジェクトを解放せずに終了した理由を示唆する追加のエラーを調べてください。

追加情報 記号定数 : SRV\_EWAITFAIL

## エラー 16361

重大度 20

メッセージ %1! で CreateEvent() が異常終了しました。

説明 CreateEvent() または comn\_create\_event() は、おそらくメモリなどのリソース不足のため、指定された API で失敗しました。

アクション	基本的なオペレーティング・システム・エラーを検索してください。UNIX では、プロセス・トレース・ユーティリティ (truss など) を使用します。Microsoft Windows では、Sybase サポート・センタに連絡して、GetLastError() を呼び出すインストールメント化されたバージョンの Sybase ライブラリを依頼してください。
追加情報	記号定数 : SRV_ECREATEEVENT

## エラー 16362

重大度	20
メッセージ	%1! で SetEvent() が異常終了しました。
説明	SetEvent() または comn_signal_event() が指定された API で失敗しました。
アクション	基本的なオペレーティング・システム・エラーを検索してください。UNIX では、プロセス・トレース・ユーティリティ (truss など) を使用します。Microsoft Windows では、Sybase サポート・センタに連絡して、GetLastError() を呼び出すインストールメント化されたバージョンの Sybase ライブラリを依頼してください。
追加情報	記号定数 : SRV_ESETEVENT

## エラー 16363

重大度	20
メッセージ	コルーチン・モードを開始することができませんでした。
説明	内部ルーチンがミューテックス・セマフォのコントロールを作成および取得しようとしています。
アクション	Sybase サポート・センタに連絡して、この致命的エラーへのサポートを依頼してください。
追加情報	記号定数 : SRV_ESTARTCOROUTINE

## エラー 16364

重大度	20
メッセージ	コルーチン・モードを終了することができませんでした。
説明	スレッド・クリーンアップが失敗しました。
アクション	Sybase サポート・センタに連絡して、この致命的エラーへのサポートを依頼してください。
追加情報	記号定数 : SRV_EENDCOROUTINE

## エラー 16365

重大度	20
メッセージ	%1! で CreateMutex() が異常終了しました。
説明	CreateMutex() (Microsoft Windows) または pthread_mutex_init() (UNIX) が指定された API でのミューテックスの作成に失敗しました。
アクション	基本的なオペレーティング・システム・エラーを検索してください。UNIX では、プロセス・トレース・ユーティリティ (truss など) を使用します。Microsoft Windows では、Sybase サポート・センタに連絡して、GetLastError() を呼び出すインストルメント化されたバージョンの Sybase ライブラリを依頼してください。
追加情報	記号定数 : SRV_ECREATEMUTEX

## エラー 16366 ( 廃止予定 )

重大度	N/A
メッセージ	%1! で mutex の削除が異常終了しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EDELETEMUTEX

## エラー 16367

重大度	10
メッセージ	%1!: CS_SERVERMSG 引数の %2! フィールドは最初の呼び出しでは %3! に設定されなければなりません。
説明	アプリケーションは、 <code>srv_sendinfo()</code> の最初の呼び出し前に <code>CS_SERVERMSG</code> ステータス・マスクを <code>CS_FIRST_CHUNK</code> に設定しませんでした。
アクション	<a href="#">「srv_sendinfo」(445 ページ)</a> および <a href="#">表 2-10 (66 ページ)</a> を参照してください。
追加情報	記号定数: <code>SRV_EBADMSGFIELD</code>

## エラー 16368

重大度	10
メッセージ	%1!: メッセージは初期化されましたが、完了しませんでした。
説明	<code>srv_senddone()</code> は、拡張エラー・データまたは情報メッセージの一部が完全に送信されていないことを検出しました。
アクション	<a href="#">「srv_sendinfo」(445 ページ)</a> 、 <a href="#">「CS_SERVERMSG 構造体」(64 ページ)</a> 、および <a href="#">表 2-10 (66 ページ)</a> を参照してください。
追加情報	記号定数: <code>SRV_ENOMSGSENT</code>

## エラー 16369

重大度	10 または 20
メッセージ	%2! でセキュリティ制御レイヤ・ルーチン %1! が異常終了しました。
説明	指定されたセキュリティ・ルーチンが、指定された API で失敗しました。
アクション	<code>truss</code> などのプロセス・トレース・ユーティリティを使用して、使用中のセキュリティ・ドライバに固有の補足エラーを検索してください。たとえば、このコマンドは、エラーに関わっている Kerberos API の詳細を次のように返します。

## エラー 16370 (廃止予定)

---

```
truss -f -o krb.out -u'a.out,libsybskrb,mech_krb5' -p  
pid
```

追加情報 記号定数: SRV\_ESCLFUNC

## エラー 16370 (廃止予定)

重大度 N/A

メッセージ セキュリティ制御レイヤの初期化に失敗しました。

説明 N/A

アクション N/A

追加情報 記号定数: SRV\_ESCLINIT

## エラー 16371

重大度 10

メッセージ セキュリティ・プロトコル・エラーが発生しました。

説明 クライアント・アプリケーションからセキュリティ・プロトコル TDS トークンを読み込んでいる間に、エラーが発生しました。

アクション このエラーを再現できる場合は、クライアント・アプリケーションとサーバ間で Ribo を実行して、どの TDS パケットのフォームが間違っているか、または交換に含まれていないかを調べてください。

追加情報 記号定数: SRV\_EBADSECPROT

## エラー 16372

重大度 10

メッセージ 認識できないセキュリティ・バージョン (%! ) を受け取りました。

説明 ログイン・フェーズのセキュリティ・ネゴシエーション時に無効なセキュリティ・バージョンが検出されました。



アクション	クライアント・アプリケーションとサーバ間で <b>Ribo</b> を実行して、 <b>TDS_MSG_SEC_OPAQUE</b> トークンに続く <b>TDS_SEC_SECSCESS</b> トークンを調べてください。 <b>TDS_SEC_SECSCESS</b> は 5 つのパラメータで構成され、その最初がタイプ <b>TDS_INTN</b> の <b>TDS</b> セキュリティ・バージョンです。この値は 50 に設定する必要があります。
追加情報	記号定数: <b>SRV_EBADSECVER</b>

## エラー 16373

重大度	10
メッセージ	不明なセキュリティ・メカニズム・オブジェクト識別子を受信しました。
説明	クライアント・アプリケーションによって要求されたセキュリティ・メカニズムはサーバにとって不明です。
アクション	<b>truss</b> などのプロセス・トレース・ユーティリティを使用して、使用中のセキュリティ・ドライバに固有の補足エラーを検索してください。たとえば、このコマンドは、エラーに関連している <b>Solaris Native Kerberos</b> および <b>Sybase Kerberos</b> 関連の API についての詳細を次のように返します。 <pre>truss -f -o krb.out -u'a.out,libsybskrb,mech_krb5' -p pid</pre>
追加情報	記号定数: <b>SRV_EBADSECMECHOID</b>

## エラー 16374

重大度	10
メッセージ	セキュリティ・セッションの有効期限切れです。
説明	セキュリティ・メカニズムのセッションが期限切れになりました。
アクション	セキュリティ・メカニズムのセッションを再初期化してください。
追加情報	記号定数: <b>SRV_ESESSTO</b>

## エラー 16375

重大度	10
メッセージ	セキュリティ・セッション・クレデンシャルの有効期限切れです。
説明	セキュリティ・メカニズムのクレデンシャルが期限切れになりました。
アクション	新しいクレデンシャル・セットを取得してください。たとえば、 <b>Kerberos</b> を使用している場合は、 <b>kinit</b> コマンドを発行して新規クレデンシャルを取得する必要があります。
追加情報	記号定数 : SRV_ECREDITO

## エラー 16376 ( 廃止予定 )

重大度	N/A
メッセージ	無効なセキュリティ・シグニチャを受信しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EBADSIGN

## エラー 16377 ( 廃止予定 )

重大度	N/A
メッセージ	セキュリティ暗号の復号に失敗しました。
説明	N/A
アクション	N/A
追加情報	記号定数 : SRV_EBADSEAL

## エラー 16378

重大度	10
メッセージ	セキュリティ・メカニズム名が長すぎます。
説明	セキュリティ・メカニズム名の長さが 32 文字 (最大値) を超えています。
アクション	セキュリティ・メカニズム名の長さが 32 文字の最大長以下であることを確認してください。
追加情報	記号定数: SRV_ESECMECHLEN

## エラー 16379

重大度	10 または 20
メッセージ	List Manager の動作が失敗しました。
説明	このエラーの重大度が 10 の場合は通常、 <i>interfaces</i> ファイル内の <i>secmech</i> エントリに関わる問題です。重大度が 20 の場合は、セキュリティ・メカニズムを格納するためのリストの割り当てに問題が生じたことを示します。
アクション	重大度が 10 の場合は、 <i>interfaces</i> ファイル内の <i>secmech</i> エントリをチェックし、それを <i>\$\$YBASE/config/objectid.dat</i> 内の <i>secmech</i> エントリと比較して有効であることを確認してください。重大度が 20 の場合は、 <i>truss</i> などのアプリケーション・トレース・ユーティリティを使用して、リソース問題 (メモリ不足など) があるか確認してください。
追加情報	記号定数: SRV_ELMAN

## エラー 16380

重大度	10
メッセージ	不正なセキュリティ・バージョンのセキュリティ・トークンを受信しました。

説明	TDS_MSG_SEC_OPAQUE メッセージ・トークンの後に続くパラメータの 1 つが、ログイン・フェーズのセキュリティ・ネゴシエーション時にエラーを検出しました。
アクション	Ribo を実行して、TDS_MSG_SEC_OPAQUE メッセージ・トークンの後に続く 5 つのパラメータを次のように調べてください。 <ul style="list-style-type: none"> <li>第 1 パラメータタイプ TDS_INTN の TDS セキュリティ・バージョン値 :50。</li> <li>第 2 パラメータタイプ TDS_INTN のセキュリティ・メッセージ値 :TDS_SEC_SECSCESS。</li> <li>第 3 パラメータタイプ使用中のセキュリティ・メカニズムのオブジェクト ID (OID) であり、このタイプは TDS_VARBINARY。</li> <li>第 4 パラメータタイプ TDS_LONGBINARY の不透明セキュリティ・トークン。</li> <li>第 5 パラメータタイプ TDS_INTN の、要求されたセキュリティ・サービス。</li> </ul>
追加情報	記号定数 : SRV_EBADSECTOK

## エラー 16381

重大度	10
メッセージ	要求されたセキュリティ・サービスはサポートされていません。
説明	セキュリティ・メカニズムがすべてのセキュリティ・サービスをサポートしているわけではありません。
アクション	指定されたサービスが現在のセキュリティ・メカニズムでサポートされているかどうか判断してください。 <i>action</i> を CS_SUPPORTED に設定し、 <i>buffer</i> を CS_BOOL 変数のアドレスに設定し、 <i>property</i> をセキュリティ・サービスを表す記号プロパティ定数として設定して、 <code>ct_config()</code> または <code>ct_con_props()</code> を呼び出してください。「 <a href="#">セキュリティ・サービス</a> 」(196 ページ) をレビューしてください。
追加情報	記号定数 : SRV_ESECSRVNOTSUP

## エラー 16382

重大度	15
メッセージ	設定された最大数のミューテックスがすでに作成済みです。
説明	<code>srv_createmutex()</code> は、許容されるミューテックスの最大数を越えたため失敗しました。
アクション	ミューテックスの数を、 <code>oscompat.h</code> で定義したミューテックスの最大数である <code>SRV_MAX_MutexES</code> に制限してください。
追加情報	記号定数: <code>SRV_EMAXMUTEXES</code>

## エラー 16383

重大度	20
メッセージ	<code>TlsAlloc()</code> は %1! で失敗しました。
説明	プロセス内のすべてのスレッドに表示可能なスレッド固有のデータ・キーを作成しようとする試み (プラットフォームに応じて <code>pthread_key_create()</code> または <code>TlsAlloc()</code> を使用) が失敗しました。このタイプの失敗は通常、オペレーティング・システムのリソース不足が原因です。
アクション	<code>truss</code> などのアプリケーション・トレース・ユーティリティを使用して、基本的なオペレーティング・システム・エラーを確認してください。
追加情報	記号定数: <code>SRV_ETLSALLOC</code>

## エラー 16384 ( 廃止予定 )

重大度	N/A
メッセージ	<code>TlsFree()</code> は %1! で失敗しました。
説明	N/A
アクション	N/A
追加情報	記号定数: <code>SRV_ETLSFREE</code>

## エラー 16385

重大度	20
メッセージ	<code>TlsSetValue()</code> は %1! で失敗しました。
説明	スレッド固有のデータ・キーを設定する試み (プラットフォームに応じて <code>TlsSetValue()</code> または <code>pthread_setspecific()</code> を使用) が失敗しました。
アクション	このタイプの失敗は通常、オペレーティング・システムのリソース不足が原因です。 <code>truss</code> などのアプリケーション・トレース・ユーティリティを使用して、基本的なオペレーティング・システム・エラーを確認してください。
追加情報	記号定数: <code>SRV_ETLSSETVAL</code>

## エラー 16386

重大度	20
メッセージ	<code>TlsGetValue()</code> は %1! で失敗しました。
説明	<code>TlsGetValue()</code> によるスレッド固有のデータ・キーを取得する試みが失敗しました。
アクション	Sybase サポート・センタに連絡して、 <code>TlsGetValue()</code> の正常終了を妨げている基本的なオペレーティング・システム・エラーを報告するために <code>GetLastError()</code> を使用するインストールメント化されたバージョンの Sybase ライブラリを依頼してください。
追加情報	記号定数: <code>SRV_ETLSGETVAL</code>

## エラー 16387 ( 廃止予定 )

重大度	N/A
メッセージ	<code>addr_trantype</code> または <code>addr_tranaddress</code> の長さが無効です。
説明	N/A
アクション	N/A
追加情報	記号定数: <code>SRV_EBADTRANADDRLEN</code>

## エラー 16388

重大度	10
メッセージ	%!!: 非同期マネージャを実行することができませんでした。
説明	サーバ名の長さ (%!!) が無効です。 <code>srv_getserverbyname()</code> に渡された <code>namelen</code> パラメータが無効です。
アクション	1 以上であり、255 より小さい <code>namelen</code> パラメータを指定してください。
追加情報	記号定数: <code>SRV_ESRVNAMELEN</code>

## エラー 16389

重大度	10
メッセージ	クライアントは HA をサポートしていません。
説明	<code>srv_send_ctlinfo()</code> は、クライアント・アプリケーションが拡張高可用性 (HA) フェイルオーバーをサポートしていないために失敗しました。
アクション	クライアント・アプリケーションとサーバ間で <code>Ribo</code> を実行して、クライアントが <code>TDS</code> レベル 5.0 またはそれ以降をインストールしていて、 <code>CS_CAP_EXTENDEDFAILOVER</code> ビットが <code>CS_TRUE</code> に設定されていることを確認してください。
追加情報	記号定数: <code>SRV_ENOSESSION</code>

## エラー 16390

重大度	10
メッセージ	%!! で始まる識別子が長すぎます。
説明	<code>srv_descfmt()</code> は、クライアント・アプリケーションが 30 文字を超える長さのテーブル名やカラム名を表すラージ ID をサポートしていないために失敗しました。

アクション	クライアント・アプリケーションとサーバ間で <b>Ribo</b> を実行して、 <b>CS_REQ_LARGEIDENT</b> 要求機能が <b>FALSE</b> に設定されているか、または <b>CS_NO_LARGEIDENT</b> が <b>TRUE</b> に設定されているかどうかを確認してください。このように設定されている場合、クライアント・アプリケーションはラージ ID をサポートしていないことを示します。
追加情報	記号定数 : <b>SRV_ENOLARGEID</b>

## エラー 16391

重大度	10
メッセージ	文字のコピーを開始する位置 (%1!) を要求バッファの文字数 (%2!) より大きくすることはできません。
説明	<code>srv_langcpy()</code> が不適切な <i>start</i> パラメータ値を受け取りました。
アクション	<i>start</i> 値が言語コマンドまたは RPC コマンドの長さよりも短くなるように、コードを訂正してください。
追加情報	記号定数 : <b>SRV_EBADSTART</b>
このエラーが発生したバージョン	12.5.1 ESD #21、15.0、およびそれ以降のバージョンです。

## エラー 16392

重大度	10
メッセージ	設定された最大数のメッセージ・キューがすでに作成済みです。
説明	<code>srv_createmsgq()</code> が呼び出されましたが、設定された最大数のメッセージ・キューがすでに使用されています。
アクション	<code>srv_deletemsgq()</code> を使用してメッセージ・キューを解放するか、メッセージ・キューの数を増やすように <b>SRV_S_NUMMSGQUEUES</b> を設定してください。
追加情報	記号定数 : <b>SRV_EMAXMQUEUES</b>



このエラーが発生したバージョン 15.0 以降

## エラー 16393

重大度 10

メッセージ %1!: プロパティ (%2!) は利用できません。

説明 `srv_thread_props(.., CS_GET, property, ..)` は、指定されたスレッドで使用できないプロパティに対して呼び出されました。現在、このエラーは、クライアント接続でセッション ID が指定されない場合に `SRV_T_SESSIONID` のみに対して発生します。このプロパティは、クライアント接続がマイグレートされたかフェイルオーバーしたクライアント・スレッドのみで使用できます。

アクション スレッドが指定されたプロパティに有効な値を持っていることが判明している場合のみ、このプロパティ値を取得してください。

追加情報 記号定数: `SRV_EPROPNOTAVAIL`

このエラーが発生したバージョン 15.0 ESD #14 以降

## エラー 16394

重大度 10

メッセージ %1!: 接続はマイグレートできません。

説明 `srv_send_ctlinfo(srvproc, SRV_CTL_MIGRATE, ..)` は、クライアント・スレッドに対して、マイグレーション不能な接続から呼び出されました。

アクション 接続をマイグレートしようとする前に、その接続が `CS_REQ_MIGRATE` を使用してマイグレート可能かどうかを確認してください。

追加情報 `srv_capability_info(.., CS_GET, CS_CAP_REQUEST, ..)` を使用して、接続がマイグレート可能かどうかをテストしてください。記号定数: `SRV_ENOTMIGRATABLE`。

このエラーが発生したバージョン 15.0 ESD #14 以降

## エラー 16395

重大度	10
メッセージ	%1!: この時点でマイグレーションをキャンセルすることはできません。
説明	マイグレーションの取り消し要求が発行されましたが、クライアント・アプリケーションはすでにマイグレーションを開始しています。このエラーは、前のマイグレーションがまだ進行中の間に新しいマイグレーションを開始したときにも発生することがあります。
アクション	マイグレーション要求を取り消してから、SRV_MIGRATE_STATE イベント・ハンドラを終了してください。
追加情報	記号定数: SRV_EMIGNOTCANCEL
このエラーが発生したバージョン	15.0 ESD #14 以降

## エラー 16396

重大度	10
メッセージ	マイグレーションが失敗しました。SRV_MIGRATE_STATE ハンドラはインストールされていません。
説明	クライアント接続がマイグレーションに失敗し、この失敗を処理するための SRV_MIGRATE_STATE イベント・ハンドラがインストールされていません。
アクション	SRV_MIGRATE_STATE ハンドラをインストールして、接続のマイグレーションが失敗したときに適切なアクションがとれるようにしてください。
追加情報	これは情報メッセージです。記号定数: SRV_EMIGFAILED.
このエラーが発生したバージョン	15.0 ESD #14 以降

## エラー 16397

重大度	10
-----	----

メッセージ	プロパティ %1! はリスナ・スレッドでのみ設定できます。
説明	指定されたプロパティを別のタイプのスレッドに設定しようとした。
アクション	指定されたプロパティをリスナ・スレッドのみに設定してください。
追加情報	記号定数 : SRV_ENOTLISTENER
このエラーが発生したバージョン	15.5 以降

## エラー 16398

重大度	10
メッセージ	最大リスナ・カウント (%1!) を超えます。
説明	すでにリスナの設定数に達したときに、別のリスナを開始しようとした。
アクション	リスナ数を増やすように SRV_S_MAXLISTENERS を設定してください。
追加情報	記号定数 : SRV_EXMAXLISTEN
このエラーが発生したバージョン	15.5 以降



# 用語解説

## Adaptive Server Enterprise

Sybase のクライアント/サーバ・アーキテクチャにおけるサーバ。Adaptive Server Enterprise は、複数のデータベースと複数のユーザを管理します。ディスク上にあるデータの実際のロケーションを監視し、論理データ記述から物理データ記憶領域へのマッピングを管理します。メモリ内のデータ・キャッシュとプロシージャ・キャッシュの保守も行います。

## 配列

複数の同じタイプの変数からなる構造体。各変数は、個々にアドレスリングされます。

## 配列バインド (array binding)

結果カラムを配列変数にバインドする処理。フェッチのときは、複数のロー分のカラムが変数にコピーされます。

## バッチ (batch)

コマンドまたは文の集まり。

Client-Library のコマンド・バッチは、`ct_send` へのアプリケーションの呼び出しで終了する、1つ以上の Client-Library のコマンドです。たとえば、アプリケーションは、カーソルに対する宣言、ローの選択、オープンを実行する複数のコマンドをまとめてバッチ処理することができます。

Transact-SQL 文バッチは、1つの Client-Library コマンドまたは Embedded SQL 文によって Adaptive Server Enterprise に送信される1つ以上の Transact-SQL 文です。

## ブラウザ・モード

DB-Library と Client-Library アプリケーションが、一度に1つのローの値を更新しながらデータベース・ローをブラウザする方法。同様の機能を果たすカーソルの方が、一般に扱いやすくなっています。

## バルク・コピー (bulk copy)

データベースからデータをコピーしたり、データベースへデータをコピーしたりするコピー・ユーティリティ。bcp と呼ばれます。

## コールバック・イベント (callback event)

Open Client と Open Server におけるコールバック・ルーチンをトリガするイベント。

## コールバック・ルーチン (callback routine)

コールバック・イベントと呼ばれるトリガ・イベントにตอบสนองして、Open Client または Open Server が呼び出すルーチン。

機能	クライアント／サーバ接続の機能が、その接続について許可されるクライアント要求とサーバ応答の種類を決定します。
文字セット (character set)	各文字をユニークに定義するコード化スキームを持つ特定の ( 通常、標準化された ) 文字の集まり。ASCII と ISO 8859-1 (Latin 1) は、よく使用される文字セットです。
文字セット変換 (character set conversion)	サーバへ入出力するときの文字セットのコード化スキームの変換。サーバとクライアントが異なる文字セットを使って通信するとき、変換が行われます。たとえば、Adaptive Server Enterprise が ISO 8859-1 を使用し、クライアントが CodePage 850 を使用する場合、文字セット変換をオンにして、サーバとクライアントが、受け渡しされるデータを同じように解釈するようにします。
クライアント (client)	クライアント／サーバ・システムにおいて、サーバへ要求を送り、この要求に対する結果に対して処理を行う部分。
Client-Library	Open Client の一部で、クライアント・アプリケーションを記述するためのルーチンの集まり。Client-Library は、Sybase 製品ラインのカーソルや他の高度な機能を取り込むように設計されたライブラリです。
コード・セット (code set)	「文字セット (character set)」を参照してください。
照合順 (collating sequence)	「ソート順 (sort order)」を参照してください。
コマンド (command)	Client-Library において、ct_command、ct_dynamic、または ct_cursor に対するアプリケーションの呼び出しで始まり、ct_send に対するアプリケーションの呼び出しで終了するサーバ要求。
コマンド構造体	(CS_COMMAND) Client-Library アプリケーションがコマンドの送信と結果の処理に使用する Client-Library の隠し構造体。
接続構造体	(CS_CONNECTION) コンテキスト内にクライアント／サーバ接続を定義する Client-Library の隠し構造体。
コンテキスト構造体	(CS_CONTEXT) Client-Library または Open Server アプリケーション内でアプリケーション「コンテキスト」または操作環境を定義する CS-Library の隠し構造体。CS-Library routines cs_ctx_alloc と cs_ctx_drop は、それぞれコンテキスト構造体の割り付けと解除を行います。
変換 (conversion)	「文字セットの変換」を参照してください。
CS-Library	Client-Library と Server-Library のアプリケーションの両方で役立つユーティリティ・ルーチンの集まり。Open Client および Open Server の両方に含まれています。

<b>現在のロー (current row)</b>	カーソルに関連して、カーソルが置かれているロー。フェッチは、カーソルに対して現在のローを取得します。
<b>カーソル (cursor)</b>	SQL 文に関連付けられた記号名。  Embedded SQL において、カーソルは複数ローのデータをホスト・プログラムに渡すデータ・セクタです。このローの受け渡しは、一度に 1 つずつ行われます。
<b>データベース (database)</b>	特別な目的のために組織化された、関連するデータ・テーブルとその他のデータベース・オブジェクトの集まり。  「スクロール可能カーソル (scrollable cursor)」も参照してください。
<b>データ型 (datatype)</b>	変数に有効な値と演算を表す定義属性。
<b>DB-Library</b>	Open Client の一部で、クライアント・アプリケーションを記述するためのルーチンの集まり。
<b>デッドロック (deadlock)</b>	データにロックを保持している 2 人のユーザが、互いに他方のデータのロックを獲得しようとしたときに起こる状態。Adaptive Server Enterprise がデッドロックを検出すると、片方のユーザのプロセスを強制終了することによってこの状態を解決します。
<b>デフォルト (default)</b>	明示的に何も指定されなかったときに、Open Client/Server 製品が使用する値、オプション、または動作。
<b>デフォルト・データベース (default database)</b>	ユーザがデータベース・サーバにログインしたとき、デフォルトで指定されるデータベース。
<b>デフォルト言語 (default language)</b>	1. アプリケーションに対して明示的にローカライゼーションの指定を行わないとき、Open Client/Server 製品が使用する言語。デフォルト言語は、ロケール・ファイルの「default」エントリにより決定されます。  2. ユーザが明示的に言語を選択しなかったとき、Adaptive Server Enterprise がメッセージとプロンプトに使用する言語。
<b>動的 SQL</b>	動的 SQL によって、Embedded SQL または Client-Library アプリケーションは、実行時に値の決まる変数を含む SQL 文を実行できます。
<b>エラー・メッセージ (error message)</b>	Open Client/Server 製品がエラー状態を検出したときに発行するメッセージ。

<b>イベント (event)</b>	Open Server アプリケーションに何らかの動作を行うよう要求するオカレンス。クライアント・コマンドおよび Open Server アプリケーション・コードの特定のコマンドは、イベントをトリガできます。イベントが発生すると、Open Server は、サーバ・アプリケーション・コード内の適切なイベント処理ルーチン、または適切なデフォルト・イベント・ハンドラのどちらか呼び出します。
<b>イベント・ハンドラ (event handler)</b>	Open Server におけるイベントを処理するルーチン。Open Server アプリケーションは、Open Server が提供するデフォルト・ハンドラを使用することができます。また、カスタマイズしたイベント・ハンドラをインストールすることもできます。
<b>公開された構造体 (exposed structure)</b>	内部が Open Client/Server プログラマに公開されている構造体。Open Client/Open Server プログラマは、公開された構造体の宣言、操作、割り付け解除を直接行うことができます。公開された構造体には、CS_DATAFMT 構造体などがあります。
<b>拡張トランザクション</b>	Embedded SQL における、複数の Embedded SQL 文からなるトランザクション。
<b>FIPS</b>	Federal Information Processing Standards (連邦情報処理標準) の略。FIPS フラグが有効なとき、Adaptive Server Enterprise および Embedded SQL プリコンパイラは、標準でない拡張された SQL 文を検出すると警告を発行します。
<b>ゲートウェイ (gateway)</b>	直接通信できないクライアントとサーバとの仲介として動作するアプリケーション。ゲートウェイ・アプリケーションは、クライアントとサーバの両方として動作します。クライアントからの要求をサーバに渡し、サーバからの結果をクライアントに返します。
<b>隠し構造体 (hidden structure)</b>	内部が Open Client/Server プログラマに対して隠されている構造体。Open Client/Server プログラマは、隠し構造体の割り付け、操作、割り付け解除を行うために、Open Client/Server ルーチンを使用しなければなりません。隠し構造体には、CS_CONTEXT 構造体などがあります。
<b>ホスト言語 (host language)</b>	アプリケーションを記述するときに使われるプログラミング言語。
<b>ホスト・プログラム (host program)</b>	Embedded SQL における、Embedded SQL コードを含むアプリケーション・プログラム。
<b>ホスト変数 (host variable)</b>	Embedded SQL における、Adaptive Server Enterprise とアプリケーション・プログラム間のデータ転送を可能にする変数。「インジケータ変数」、「入力変数」、「出力変数」、「結果変数」、「ステータス変数」を参照してください。



<b>インジケータ変数 (indicator variable)</b>	他の変数の値やフェッチしたデータについての特別な条件を示す変数。 Embedded SQL ホスト変数とともに使用すると、データベースの値が NULL である箇所を示します。
<b>入力変数 (input variable)</b>	情報をルーチン、ストアド・プロシージャ、または Adaptive Server Enterprise に渡すときに使う変数。
<b>interfaces ファイル (interfaces file)</b>	サーバ名をトランスポート・アドレスにマップするファイル。クライアント・アプリケーションが、サーバに接続するために <code>ct_connect</code> または <code>dbopen</code> を呼び出すと、Client-Library または DB-Library が <code>interfaces</code> ファイルからサーバのアドレスを検索します。ただし、すべてのプラットフォームが <code>interfaces</code> ファイルを使うわけではありません。 <code>interfaces</code> ファイルを使わないプラットフォームでは、別のメカニズムでクライアントにサーバ・アドレスを知らせます。
<b>isql スクリプト・ファイル (isql script file)</b>	Embedded SQL において、プリコンパイラが生成できる3つのファイルのうちの1つ。isql スクリプト・ファイルには、プリコンパイラが生成したストアド・プロシージャが含まれます。ストアド・プロシージャは、Transact-SQL で記述されます。
<b>キー (key)</b>	ローをユニークに識別するロー・データのサブセット。キー・データは、オープンされたカーソル内の「 <b>現在のロー</b> 」をユニークに記述します。
<b>キーワード (keyword)</b>	Transact-SQL または Embedded SQL で排他的に利用するように予約されているワードまたはフレーズ。予約語とも呼ばれます。
<b>リスティング・ファイル (listing file)</b>	Embedded SQL において、プリコンパイラが生成できる3つのファイルのうちの1つ。リスティング・ファイルには、入力ファイルのソース文と、情報、警告、エラーなどのメッセージが含まれます。
<b>ロケール・ファイル</b>	ロケール名を言語と文字セットのペアにマッピングするファイル。Open Client/Server 製品は、ローライゼーション情報をロードするときにこのロケール・ファイルを調べます。
<b>ロケール名</b>	言語と文字セットのペアを表す文字列。ロケール名は、ロケール・ファイルにリストされています。Sybase があらかじめ定義しているロケール名の他に、システム管理者が別のロケール名を定義し、ロケール・ファイルに追加することもできます。

**ロケール構造体  
(locale structure)**

(Client-Library または Open Server アプリケーションのためのカスタム・ローカライゼーション値を定義する CS-Library の隠し構造体 (CS\_LOCALE)。アプリケーションは、CS\_LOCALE を使用して、使用される言語、文字セット、日付順、ソート順を定義できます。ロケール構造体の割り付けと割り付け解除には、CS-Library ルーチン cs\_loc\_alloc および cs\_loc\_drop を使用します。

**ローカライゼーション  
(localization)**

アプリケーションを特定の言語環境で使用するために設定する処理のこと。ローカライズされたアプリケーションは、通常、各国の言語と文字セットでメッセージを作成し、その国の日時表記フォーマットを使用します。

**ログイン名**

ユーザが、サーバにログインするとき使用する名前。Adaptive Server Enterprise のログイン名が有効になるのは、Adaptive Server Enterprise がシステム・テーブル syslogins にそのユーザのエントリを持つ場合です。

**メッセージ番号  
(message number)**

エラー・メッセージをユニークに識別する番号。

**メッセージ・キュー  
(message queue)**

Open Server において、スレッドが通信するとき使用するメッセージ・ポインタのリンク・リスト。スレッドは、キューにメッセージを書き込んだり、キューからメッセージを読み込んだりすることができます。

**マルチバイト文字セット  
(multibyte character set)**

複数のバイトを使用してコード化された文字を含む文字セット。マルチバイト文字セットには、EUC JIS、シフト JIS などがあります。

**ミューテックス  
(mutex)**

相互排他セマフォ。Open Server アプリケーションが、共有オブジェクトへ排他アクセスをするために使用する論理オブジェクトのことです。

**NULL**

NULL は、明示的に割り当てられた値を持ちません。NULL は 0 でもブランクでもありません。NULL の値は、他の値と比べて大きいとも、小さいとも、同じであるともみなされません。他の NULL と比べても同じです。

**Open Server**

カスタム・サーバを作成するためのツールとインタフェースを提供する Sybase 製品。

**Open Server アプリケーション**

Open Server で構築されたカスタム・サーバ。

**出力変数 (output variable)**

Embedded SQL において、ストアド・プロシージャからアプリケーション・プログラムにデータを渡す変数。

**パラメータ  
(parameter)**

1. データをルーチンに渡すとき、およびルーチンからデータを取得するときに使用する変数。
2. ストアド・プロシージャへの引数。

<b>パススルー・モード (passthrough mode)</b>	ゲートウェイ・アプリケーションに関する 1 つの状態。 パススルー・モードのとき、ゲートウェイは、クライアントとリモート・データ・ソース間の TDS (Tabular Data Stream) パケットを、そのパケットの内容をアンパックすることなく中継します。
<b>プロパティ (property)</b>	構造体に格納される名前付きの値。コンテキスト構造体、接続構造体、スレッド構造体、コマンド構造体は、プロパティを持ちます。構造体のプロパティは、構造体の動作を決めます。
<b>クエリ (query)</b>	1. データの検索要求。通常は <code>select</code> 文です。 2. データを操作する任意の SQL 文。
<b>レジスタード・プロシージャ (registered procedure)</b>	Open Server において、名前を付けて保管される C で記述された文の集まり。Open Server が提供するレジスタード・プロシージャは、「 <b>システム・レジスタード・プロシージャ (system registered procedure)</b> 」と呼ばれます。
<b>リモート・プロシージャ・コール</b>	1. クライアント・アプリケーションが Adaptive Server Enterprise ストアド・プロシージャを実行する 2 つの方法のうちの 1 つ (もう 1 つの方法では、Transact-SQL の <code>execute</code> 文を使用します)。Client-Library のアプリケーションは、 <code>ct_command</code> を呼び出すことによって、リモート・プロシージャ・コール・コマンドを開始します。DB-Library アプリケーションは、 <code>dbrpcinit</code> を呼び出すことによって、リモート・プロシージャ・コール・コマンドを開始します。 2. クライアントが Open Server アプリケーションを使って利用できる要求のタイプの 1 つ。これに回答して Open Server は、対応するレジスタード・プロシージャを実行するか、または Open Server アプリケーションの RPC イベント・ハンドラを呼び出します。 3. ユーザが接続しているサーバとは異なるサーバ上で実行される「 <b>ストアド・プロシージャ</b> 」。
<b>結果変数 (result variable)</b>	Embedded SQL において、 <code>select</code> または <code>fetch</code> 文の結果を受け取る変数。
<b>スクロール可能カーソル (scrollable cursor)</b>	現在のカーソル位置を、カーソル結果セットの任意の場所に設定できるようにします。「 <b>カーソル (cursor)</b> 」も参照してください。
<b>サーバ (server)</b>	クライアント/サーバ・システムにおけるクライアント要求を処理し、結果をクライアントに返す部分。
<b>Server-Library</b>	Open Server アプリケーションの記述に使用するルーチンの集まり。
<b>ソート順 (sort order)</b>	文字データをソートするときの順序の決定に使用されます。「 <b>照合順</b> 」とも呼ばれます。

**SQLCA**

1. Embedded SQL アプリケーションにおいて、Adaptive Server Enterprise とアプリケーション・プログラム間の通信パスを提供する構造体。Adaptive Server Enterprise は、各 SQL 文を実行したあと、SQLCA 内のリターン・コードを格納します。
2. Client-Library アプリケーションにおいて、アプリケーションが、Client-Library およびサーバのエラー・メッセージと情報メッセージを取得するのに使用する構造体。

**sqlcode**

1. Embedded SQL アプリケーションにおいて、Adaptive Server Enterprise とアプリケーション・プログラム間の通信パスを提供する構造体。Adaptive Server Enterprise は、各 SQL 文を実行したあと、SQLCODE 内のリターン・コードを格納します。SQLCODE は、独立して存在することも、SQLCA 構造体の変数になることもできます。
2. Client-Library アプリケーションにおいて、アプリケーションが Client-Library およびサーバのエラー・メッセージと情報メッセージ・コードを取得するのに使用する構造体。

**文 (statement)**

Transact-SQL または Embedded SQL におけるキーワードで始まる命令。キーワード名は、基本オペレーションまたは実行するコマンドを表します。

**ステータス変数  
(status variable)**

Embedded SQL において、ストアド・プロシージャのリターン・ステータス値を受け取ることによって、プロシージャの成功または失敗を示す変数。

**ストアド・プロシ  
ージャ (stored  
procedure)**

Adaptive Server Enterprise における、名前を付けて保管された SQL 文とオプションのフロー制御文の集まり。Adaptive Server Enterprise が提供するストアド・プロシージャは、「**システム・プロシージャ (system procedure)**」と呼ばれます。

**システム管理者  
(System  
Administrator)**

ユーザ・アカウントの作成、パーミッションの割り当て、および新しいデータベースの作成を含むサーバ・システム管理を担当するユーザ。Adaptive Server Enterprise では、システム管理者のログイン名は“sa”です。

**システム記述子  
(system descriptor)**

Embedded SQL において、動的 SQL 文で使われる変数の記述を保持するメモリの領域。

**システム・プロシ  
ージャ (system  
procedures)**

Adaptive Server Enterprise が、システム管理のために提供するストアド・プロシージャ。これらのプロシージャは、システム・テーブルからの情報の取得を簡単にし、データベース管理とシステム・テーブルの更新などを可能にします。

<b>システム・レジスタード・プロシージャ (system registered procedures)</b>	Open Server が、レジスタード・プロシージャのノーティフィケーション (通知) の管理とステータスの監視のために提供する内部レジスタード・プロシージャ。
<b>ターゲット・ファイル (target file)</b>	Embedded SQL において、プリコンパイラが生成できる 3 つのファイル。ターゲット・ファイルは元の入力ファイルと似ていますが、すべての SQL 文が Client-Library の関数呼び出しに変換されています。
<b>TDS</b>	Sybase のクライアントとサーバが通信に使用するアプリケーション・レベルのプロトコル。TDS は、コマンドと結果を記述します。
<b>スレッド (thread)</b>	Open Server アプリケーションからライブラリ・コードまでの実行のパス。また、スタック領域、ステータス情報およびイベント・ハンドラに対応するパス。
<b>Transact-SQL</b>	データベース言語 SQL の機能拡張バージョン。アプリケーションは、Transact-SQL を使用して、Adaptive Server Enterprise と通信できます。
<b>トランザクション (transaction)</b>	バックアップまたはリカバリのために 1 つの単位として扱われる、1 つ以上のサーバ・コマンド。トランザクション内のコマンドは、1 つのグループとしてコミットされます。したがって、すべてのコマンドがコミットされるか、すべてロールバックされるかのどちらかとなります。
<b>トランザクション・モード (transaction mode)</b>	Adaptive Server Enterprise がトランザクションを管理する方法。Adaptive Server Enterprise は、2 つのトランザクション・モードをサポートしています。Transact-SQL モード (「非連鎖トランザクション」とも呼ばれる) と ANSI モード (「連鎖トランザクション」とも呼ばれる) です。
<b>ユーザ名</b>	「 <b>ログイン名 (login name)</b> 」を参照してください。



# 索引

## 記号

@@textsize グローバル変数 143

## A

ANSI 準拠、更新と削除 139

ASCII 文字フォーマット 178

## B

binary データ型 26, 29, 231

bit データ型 26, 29, 232

boundary データ型 26, 29, 240

## C

character データ型 26, 29, 233, 234

Client-Library

    srv\_thread\_props によるクライアントのパー  
    ジョンの取得 169

    コンテキスト・プロパティ 158

CS\_ABSOLUTE フェッチ・タイプ 72

CS\_ACK 動的オペレーション 309

CS\_ALL\_CAPS 引数 36

CS\_BIGDATETIME データ型 230, 234, 235, 236

CS\_BIGINT データ型 237

CS\_BIGTIME データ型 230, 234, 235, 236

CS\_BINARY データ型 229, 231

CS\_BIT データ型 229, 233

CS\_BOUNDARY\_TYPE 値 240

CS\_BROWSEDESC 構造体 41, 56

cs\_calc ルーチン 231

CS\_CANBENULL 値 60, 269

CS\_CANCEL\_ATTN 引数 21

CS\_CAP\_REQUEST 引数 35

CS\_CAP\_RESPONSE 機能 279

CS\_RES\_NOXNLMETADATA 30

CS\_CAP\_TYPE 構造体 36

CS\_CHAR データ型 229, 233

CS\_CLR\_CAPMASK マクロ 36, 37

cs\_cmp ルーチン 231

cs\_config コマンド 64, 114, 117, 158

CS\_CONNECTION 構造体 148

CS\_CONTEXT 構造体 8, 63, 115, 158

cs\_convert コマンド 115

    CS\_DATAFMT 構造体 57

cs\_convert ルーチン 231

cs\_ctx\_alloc コマンド 117

CS\_CUR\_RELOCKS\_ONCLOSE 84

CS\_CURSOR\_CLOSE コマンド 75, 81

CS\_CURSOR\_DECLARE コマンド 71, 75, 78

CS\_CURSOR\_DELETE コマンド 72, 75, 81

CS\_CURSOR\_FETCH コマンド 71, 75, 80

CS\_CURSOR\_INFO コマンド 71, 76, 79

CS\_CURSOR\_OPEN 値 76, 80

CS\_CURSOR\_UPDATE コマンド 72, 77, 81

CS\_CURSTAT\_CLOSED 値 74

CS\_CURSTAT\_DEALLOC 値 74

CS\_CURSTAT\_DECLARED 値 74

CS\_CURSTAT\_OPEN 値 74, 84

CS\_CURSTAT\_RDONLY 値 74

CS\_CURSTAT\_ROWcnt 値 74, 84

CS\_CURSTAT\_UPDATABLE 値 74

CS\_DATA\_LBIN 機能 232

CS\_DATA\_LCHAR 機能 233

CS\_DATAampfmt 構造体 305

CS\_DATAFMT 構造体 56, 60, 155

CS\_DATE データ型 230, 234

CS\_DATETIME データ型 230, 234

CS\_DATETIME4 データ型 230, 234, 235

CS\_DEALLOC 値 95

CS\_DEALLOC 動的オペレーション 309

CS\_DECIMAL データ型 230, 238

CS\_DEF\_PREC 値 59, 239

CS\_DEF\_SCALE 値 59, 239

CS\_DESCIN 値 60, 92

- CS\_DESCOUT 値 60, 93  
CS\_DESCRIBE\_INPUT 値 92  
CS\_DESCRIBE\_INPUT 動的オペレーション 309  
CS\_DESCRIBE\_OUTPUT 値 93  
CS\_DESCRIBE\_OUTPUT 動的オペレーション 309  
cs\_dt\_crack ルーチン 231, 235  
cs\_dt\_info ルーチン 231  
CS\_EXEC\_IMMEDIATE 値 94  
CS\_EXECUTE 値 94  
CS\_EXEC\_IMMEDIATE 動的オペレーション 309  
CS\_EXECUTE 動的オペレーション 309  
CS\_EXPRESSION 引数 56  
CS\_FIRST フェッチ・タイプ 72  
CS\_FIRST\_CHUNK 引数 39, 66  
CS\_FLOAT データ型 230, 238  
CS\_FMT\_NULLTERM 引数 59  
CS\_FMT\_PADBLANK 引数 59  
CS\_FMT\_PADNULL 引数 59  
CS\_FMT\_UNUSED 引数 59  
CS\_FOR\_UPDATE 値 82  
CS\_GOODDATA 値 251, 265  
CS\_HASEED ビット 40, 66  
CS\_HIDDEN 値 60  
CS\_IMAGE データ型 231, 241  
CS\_IMAGE\_TYPE 値 61  
CS\_IMAGELOCATOR データ型 230  
CS\_INPUTVALUE 値 60  
CS\_INT データ型 230, 237  
CS\_IODATA 値 61  
CS\_IODESC 構造体 60, 62, 226  
CS\_KEY 値 60, 82  
CS\_LANG\_CMD 値 120  
CS\_LAST フェッチ・タイプ 72  
CS\_LAST\_CHUNK 引数 39, 66  
CS\_LC\_ALL 値 114  
cs\_loc\_alloc コマンド 114, 117  
cs\_loc\_drop コマンド 114, 117  
CS\_LOC\_PROP 値 114, 117  
CS\_LOCALE 構造体 60, 171  
cs\_locale コマンド 114, 117  
CS\_LOCATOR データ型 239  
CS\_LOGINFORM 構造体 148, 325  
CS\_LONGBINARY データ型 229, 232  
CS\_LONGCHAR データ型 229, 233  
CS\_MAX\_MSG 引数 39  
CS\_MAX\_PREC 値 59, 239  
CS\_MAX\_SCALE 値 59, 239  
CS\_MIN\_PREC 値 59, 239  
CS\_MIN\_SCALE 値 59, 239  
CS\_MONEY データ型 231, 240  
CS\_MONEY4 データ型 231, 240  
CS\_NEXT フェッチ・タイプ 72  
CS\_NOAPICHK 値 160  
CS\_NODEFAULT 値 60  
CS\_NULLDATA 値 265  
CS\_NUMERIC データ型 230, 238  
CS\_OP\_AVG 演算子タイプ 256  
CS\_OP\_COUNT 演算子タイプ 256  
CS\_OP\_MAX 演算子タイプ 256  
CS\_OP\_MIN 演算子タイプ 256  
CS\_OP\_SUM 演算子タイプ 256  
CS\_OPT\_ANSINULL サーバ・オプション 139  
CS\_OPT\_ANSIPERM サーバ・オプション 139  
CS\_OPT\_ARITHABORT サーバ・オプション 140  
CS\_OPT\_ARITHIGNORE  
サーバ・オプション 140  
CS\_OPT\_AUTHOFF サーバ・オプション 140  
CS\_OPT\_AUTHON サーバ・オプション 140  
CS\_OPT\_CHAINXACTS サーバ・オプション 140  
CS\_OPT\_CURCLOSEONXACT  
サーバ・オプション 140  
CS\_OPT\_CURREAD サーバ・オプション 140  
CS\_OPT\_CURWRITE サーバ・オプション 140  
CS\_OPT\_DATEFIRST サーバ・オプション 140  
CS\_OPT\_DATEFORMAT  
サーバ・オプション 141  
CS\_OPT\_FIPSFLAG サーバ・オプション 141  
CS\_OPT\_FORCEPLAN サーバ・オプション 141  
CS\_OPT\_FORMATONLY  
サーバ・オプション 141  
CS\_OPT\_GETDATA サーバ・オプション 141  
CS\_OPT\_IDENTITYOFF サーバ・オプション 141  
CS\_OPT\_IDENTITYON サーバ・オプション 141  
CS\_OPT\_ISOLATION サーバ・オプション 141  
CS\_OPT\_LEVEL1 値 141  
CS\_OPT\_NOCOUNT  
サーバ・オプション 139, 141  
CS\_OPT\_NOEXEC サーバ・オプション 141  
CS\_OPT\_PARSEONLY サーバ・オプション 142  
CS\_OPT\_QUOTED\_IDENT  
サーバ・オプション 142  
CS\_OPT\_RESTRICES サーバ・オプション 142



- CS\_OPT\_ROWCOUNT サーバ・オプション 142  
 CS\_OPT\_SHOWPLAN サーバ・オプション 142  
 CS\_OPT\_STATS\_IO サーバ・オプション 142  
 CS\_OPT\_STATS\_TIME サーバ・オプション 142  
 CS\_OPT\_STR\_RTRUNC  
     サーバ・オプション 142  
 CS\_OPT\_TEXTSIZE サーバ・オプション 143  
 CS\_OPT\_TRUNCIGNORE  
     サーバ・オプション 143  
 CS\_PASSTHRU\_MORE 値 149  
 CS\_PREPARE 値 92  
 CS\_PREPARE 動的オペレーション 309  
 CS\_PREV フェッチ・タイプ 72  
 CS\_REAL データ型 230, 238  
 CS\_RELATIVE フェッチ・タイプ 72  
 CS\_RENAMED 引数 56  
 CS\_REQ\_MIGRATE 43  
 CS\_REQUEST 機能 32  
 CS\_RES\_NOTDSDEBUG 機能 30  
 CS\_RESPONSE 機能 34  
 CS\_RESPONSE\_CAP 引数 36  
 CS\_RETURN 値 60  
 CS\_SECSSESSION\_CB 値 219  
 CS\_SENSITIVITY\_TYPE 値 240  
 CS\_SERVERMSG 構造体 38, 64, 66  
     CS\_HASEED ビット 40  
 CS\_SET\_CAPMASK マクロ 36, 37  
 CS\_SMALLINT データ型 230, 237  
 CS\_SRC\_VALUE 引数 59  
 CS\_SYB\_CHARSET 値 117  
 CS\_TEXT データ型 231, 241  
 CS\_TEXT\_TYPE 値 61  
 CS\_TEXTLOCATOR データ型 230  
 CS\_TIME データ型 230, 234  
 CS\_TIMESTAMP 値 60  
 CS\_TINYINT データ型 230, 237  
 CS\_TST\_CAPMASK マクロ 36, 37  
 CS\_UBIGINT データ型 237  
 CS\_UINT データ型 237  
 CS\_UNICHAR データ型 229, 234  
 CS\_UNITEXT データ型 241  
 CS\_UNITEXTLOCATOR データ型 230  
 CS\_UPDATABLE 値 60  
 CS\_UPDATECOL 値 60  
 CS\_USER\_MAX\_MSGID 値 86  
 CS\_USER\_MSGID 値 86  
 CS\_USMALLINT データ型 237  
 CS\_VARBINARY データ型 229, 232  
 CS\_VARCHAR データ型 229, 231, 233  
 CS\_VERSION\_KEY 値 60  
 CS\_XML データ型 229  
 CS-Library 62, 64  
     エラー 64, 159  
     エラー・メッセージ 113, 115  
     コンテキスト・プロパティ 158  
     定義 7, 62  
 ct\_cancel コマンド 103  
 ct\_capability コマンド 36  
 ct\_close コマンド 104  
 ct\_command コマンド 21, 86, 105, 120  
 ct\_connect コマンド 36, 103  
 ct\_cursor コマンド 68  
 ct\_exit コマンド 104  
 ct\_getloginfo コマンド 148  
 ct\_recvpassthru コマンド 149  
 ct\_send コマンド 105  
 ct\_sendpassthru コマンド 149  
 ct\_setloginfo 148  
 curemd フィールド、SRV\_CURDESC  
     構造体 74, 84  
 curid フィールド、SRV\_CURDESC 構造体 78  
 curstatus フィールド  
     SRV\_CURDESC 構造体 73
- ## D
- datetime 型  
     CS\_DATE 234  
     CS\_TIME 234  
 datetime データ型 26, 29, 234  
     8 バイトへの変換 169  
 dbcancel コマンド 20  
 decimal データ型 26, 29, 59  
 DSLISTEN 環境変数 338
- ## E
- EBCDIC 文字フォーマット 178  
 execute 文 67

**F**

floating point データ型 26, 29  
8 バイトへの変換 169  
表現 170

**I**

I/O 記述子構造体 61  
I/O チャンネル  
スレッド 171  
identity カラム 141  
image データ 61  
転送 178  
image データ型 26, 225  
srv\_get\_data 320  
srv\_get\_text 321  
integer 型 26, 29, 237  
interfaces ファイル 211  
srv\_props による名前の指定 161  
サーバ名の検索 165  
ディレクトリ・サービス 89  
isbrowse 構造体要素 56

**L**

libtcl.cfg ファイル 88

**M**

malloc C ルーチン 250  
money データ型 27, 29, 240  
8 バイトへの変換 169

**N**

Net-Library  
ネットワーク・サービスの提供 7  
Net-Library トレース・ファイル  
srv\_props による指定 162  
NULL 139

null 139  
NULL が許可される bit データ型 26

**O****Open Server**

クライアント/サーバ・アーキテクチャにおける位置付け 4  
ヘッダ・ファイル 7

**Open Server アプリケーション**

Adaptive Server Enterprise との比較 3  
ゲートウェイ 5  
初期化 9  
スタンドアロン 4  
簡単なプログラム 9, 12  
定義 3  
補助 4

**Open Server アプリケーションの構築 7, 18**

oserror.h ヘッダ・ファイル 98  
ospublic.h ヘッダ・ファイル 132

**P**

precision  
decimal データ型 59, 239

**S**

scale  
decimal データ型 59, 239  
security データ型 240  
select クエリ・オプション 141  
select 文 142  
sensitivity データ型 27, 29, 240  
Server-Library  
コンテキスト・プロパティ 159  
server-Library  
バージョン 166  
set コマンド 138  
SIGTRAP シグナル 295  
sp\_ps 189, 505, 508  
sp\_regcreate 508

- sp\_regdrop 515
- sp\_reglist 516, 517
- sp\_regnowatch 517, 518
- sp\_regwatch 518, 520
- sp\_regwatchlist 520
- sp\_serverinfo 119, 520, 521
  - sp\_serverinfo 要求への応答 119
- sp\_terminate 189, 521, 523
- sp\_who 189, 523, 525
- SQL クエリ 120
- srv\_alloc 248, 250
- srv\_alt\_bind 250, 255, 258, 263
- srv\_alt\_descampfmt 254, 255, 258, 263
- srv\_alt\_header 254, 258, 259, 263
- srv\_alt\_xferdata 254, 258, 261, 263
- SRV\_APPDEFINED 値 172
- SRV\_ATTENTION イベント 20, 103, 134, 160
- SRV\_ATTENTION
  - イベント・ハンドラ 20, 22, 161
  - クライアント切断の処理  
のための呼び出し 22
- SRV\_BIG\_ENDIAN 値 169
- srv\_bind 152, 155, 263, 266
  - CS\_DATAFMT 構造体 57
- srv\_bmove 269, 271
- SRV\_BULK イベント 101, 103, 178, 226
- SRV\_BULKLOAD 値 178
- SRV\_TEXTLOAD 値 178
- srv\_bzero 271, 272
- SRV\_C\_DEBUG 機能 277
- SRV\_C\_DEFAULTPRI 定数 132
- SRV\_C\_EXIT 機能 277
- SRV\_C\_EXIT コールバック・タイプ 273
- SRV\_C\_EXIT ステータスの移行 133
- SRV\_C\_LOWPRIORITY 定数 132
- SRV\_C\_MAXPRIORITY 定数 132
- SRV\_C\_MQUEUE 値 331
- SRV\_C\_Mutex 値 331
- SRV\_C\_PREEMPT 機能 277
- SRV\_C\_PROCEXEC コールバック・タイプ 273
- SRV\_C\_PROCEXEC ステータスの移行 133
- SRV\_C\_RESUME 機能 277
- SRV\_C\_RESUME コールバック・タイプ 273
- SRV\_C\_RESUME ステータスの移行 133
- SRV\_C\_SELECT 機能 277
- SRV\_C\_SUSPEND 機能 277
- SRV\_C\_SUSPEND コールバック・タイプ 273
- SRV\_C\_SUSPEND コールバック・ハンドラ 132
- SRV\_C\_SUSPEND ステータスの移行 133
- SRV\_C\_TIMESLICE 機能 277
- SRV\_C\_TIMESLICE コールバック・タイプ 273
- SRV\_C\_TIMESLICE ステータスの移行 133
- srv\_callback 272, 276
  - マルチスレッド・プログラミング 132, 133
- srv\_capability 129, 276, 277
- srv\_capability\_info 25, 35, 36, 278, 279
  - イベント・ハンドラ 37
- SRV\_CHALLENGE 値 172
- SRV\_CHAR\_ASCII 値 178
- SRV\_CHAR\_EBCDIC 値 178
- SRV\_CHAR\_UNKNOWN 値 178
- SRV\_CLEAROPTION 値 139
- SRV\_CLIENT ログイン・タイプ 181
- SRV\_CONNECT イベント 101, 103, 126
- SRV\_CONNECT イベント・ハンドラ 31, 35, 37, 135, 137, 153, 176, 213
  - srv\_getlogininfo 325
  - セキュリティ・セッション 214, 219
  - パススルー・モード 147
- SRV\_CONTINUE 戻り値 275
- srv\_createmsgq 131, 282, 284
- srv\_createmutex 286
- srv\_createproc 286, 288
- SRV\_CTL\_MIGRATE 43
- SRV\_CUR\_ASKSTATUS 値 76
- SRV\_CUR\_DEALLOC 値 75, 85
- SRV\_CUR\_DYNAMIC 値 75
- SRV\_CUR\_HASARGS 値 76
- SRV\_CUR\_INFORMSTATUS 値 76
- SRV\_CUR\_RDONLY 値 75
- SRV\_CUR\_SETROWS 値 76
- SRV\_CUR\_UNUSED 値 75, 76, 85
- SRV\_CUR\_UPDATABLE 値 75, 85
- SRV\_CURDATA 値 152
- SRV\_CURDATA データ型 264
- SRV\_CURDESC 構造体 69, 73, 290
  - curcmd フィールド 74, 84
  - curid フィールド 78
  - curstatus フィールド 73
- SRV\_CURSOR イベント 77, 103
- SRV\_CURSOR イベント・ハンドラ 68, 77, 153, 290

- srv\_cursor\_props 69, 79, 288, 291, 292
- srv\_dbg\_stack 291, 292, 294
- srv\_dbg\_switch 294, 295
- SRV\_DEBUG 戻り値 275
- srv\_define\_event 108, 295, 297
- srv\_deletemsgq 131, 297, 300
- srv\_deletemutex 300, 301
- srv\_descampfmt 301, 306
- srv\_descfmt 30, 152, 155
  - CS\_DATAFMT 構造体 57
  - SRV\_CURDATA 引数 78
  - SRV\_UPCOLDATA 引数 78
- SRV\_DISCONNECT イベント 101, 104, 160, 314, 317
  - 致命的なエラー 99
- SRV\_DISCONNECT
  - イベント・ハンドラ 22, 108, 169
- SRV\_DS\_PROVIDER プロパティ 88, 209
- SRV\_DYN\_values 307
- srv\_dynamic 91, 306, 310
- SRV\_DYNAMIC イベント 104, 309
- SRV\_DYNAMIC イベント・ハンドラ 91, 153
- SRV\_DYNAMICDATA データ型 264
- SRV\_DYNDATA 値 92, 93
- SRV\_ENCRYPT 値 172
- SRV\_ENO\_OS\_ERR 値 99
- srv\_envchange 310, 311
- SRV\_EQUEUED イベント・タイプ 296
- SRV\_ERRORDATA データ型 264
- SRV\_ERRORDATA 引数 41
- srv\_event 101, 104, 107, 108, 297, 311, 315
- srv\_event\_deferred 21, 108, 315, 318
- SRV\_FATAL\_PROCESS エラー重大度 99
- SRV\_FATAL\_SERVER エラー重大度 99
- SRV\_FLT\_浮動小数点フォーマット 180
- srv\_free 250, 318, 319
- srv\_freesserveraddrs 319
- srv\_get\_text 226, 319, 324
- srv\_getloginfo 31, 148, 324, 326
- srv\_getmsgq 129, 131, 326, 328
- srv\_getobjid 328, 331
- srv\_getobjname 284, 286, 331, 334
- SRV\_GETOPTION 値 139
- srv\_getserverbyname 334
- srv\_handle 101, 334, 338
- SRV\_HASPARAMS 値 87
- SRV\_I\_DELETED 値 327
- SRV\_I\_INTERRUPTED 値 327
- SRV\_I\_NOEXIST 値 329
- SRV\_I\_PASSTHRU\_MORE 値 149
- SRV\_I\_UNKNOWN 値 327, 329
- SRV\_I\_WOULDWAIT 値 327
- SRV\_IMAGELOAD 値 178
- SRV\_INFO エラー重大度 99
- srv\_init 338, 340
  - ディレクトリ・サービス 89
- SRV\_KEYDATA データ型 264
- srv\_langcpy 121, 340, 343
- SRV\_LANGDATA 型 264
- srv\_langlen 121, 343, 344
- SRV\_LANGUAGE イベント 105, 120
- SRV\_LANGUAGE
  - イベント・ハンドラ 109, 120, 153
  - オプション要求 138
  - 動作の再ネゴシエーション 137
- SRV\_LISTENER\_POSTBIND
  - イベント 96, 105, 471
- SRV\_LISTENER\_PREBIND イベント 95, 105, 471
- SRV\_LITTLE\_ENDIAN 値 169
- srv\_lockmutex 129, 344, 347
- srv\_log 98, 347, 349
- SRV\_M\_NOWAIT 値 327
- SRV\_M\_READ\_ONLY 値 327
- SRV\_M\_WAIT 値 327
- SRV\_M\_WAKE\_INTR 20
- srv\_mask 349, 350
- SRV\_MAXRESMSG メッセージ ID 86
- SRV\_MIG\_STATE 列挙型 48
- SRV\_MIGRATE\_RESUME 45
- SRV\_MIGRATE\_RESUME イベント 106
- SRV\_MIGRATE\_STATE 46
- SRV\_MINRESMSG メッセージ ID 86
- srv\_msg 87, 350, 353
- SRV\_MSG イベント 86, 106
- SRV\_MSG イベント・ハンドラ 102, 153
- SRV\_MSGDATA データ型 264, 268
- SRV\_S\_MSGPOOL プロパティ 162
- SRV\_NEGDATA データ型 264
- srv\_negotiate 137, 355, 363
- SRV\_NOPARAMS 値 87
- srv\_numparams 153, 193, 363, 365
- SRV\_OPTION イベント 105, 106, 138
- SRV\_OPTION イベント・ハンドラ

- 動作の再ネゴシエーション 137
- srv\_options 138, 365, 372
- srv\_orderby 372
- srv\_poll (UNIX のみ) 374, 377
- SRV\_POLL 機能 277
- SRV\_PROC 構造体 103
- SRV\_PROCLIST 構造体 189
- srv\_props 21, 158, 377, 378
- srv\_putmsgq 131, 384, 387
- srv\_realloc 387, 388
- srv\_recvpassthru 149, 388, 391
- srv\_regcreate 187, 391, 393
- srv\_regdefine 187, 393, 396
- srv\_regdrop 187, 396, 398
- srv\_regexec 188, 398, 400
- srv\_reginit 188, 400, 403
- srv\_reglist 188, 403, 404
- srv\_reglistfree 188, 404, 406
- srv\_regnowatch 188, 406, 408
- srv\_regparam 187, 408, 411
- srv\_regwatch 188, 411, 414
- srv\_regwatchlist 188, 414, 416
- SRV\_ROWDATA 値 152
- SRV\_ROWDATA データ型 264
- SRV\_RPC イベント 106, 192
  - レジスタード・プロシージャ 185
- SRV\_RPC イベント・ハンドラ 153, 185, 192
  - エラーのトラップ 193
- SRV\_RPCDATA データ型 264
- srv\_rpcdb 193, 416, 418
- srv\_rpcname 193, 418, 421
- srv\_rpcnumber 193, 421, 422
- srv\_rpcoptions 422, 424
- srv\_rpcowner 193, 424, 426
- srv\_run 107, 426, 427
- SRV\_S\_ALLOCFUNC プロパティ 160
- SRV\_S\_APICLK プロパティ 160
- SRV\_S\_ATTREASON プロパティ 160
- SRV\_S\_CURTHREAD プロパティ 160
- SRV\_S\_DEFQUEUE SIZE プロパティ 161
- SRV\_S\_DISCONNECT プロパティ 22, 161
- SRV\_S\_DS\_REGISTER プロパティ 89, 161
- SRV\_S\_DS\_PROVIDER プロパティ 161
- SRV\_S\_ERRHANDLE プロパティ 97, 161
- SRV\_S\_FREEFUNC プロパティ 161
- SRV\_S\_IFILE プロパティ 161
- SRV\_S\_INHIBIT プロパティ 190
- SRV\_S\_INHIBIT 戻り値 275
- SRV\_S\_LOGFILE プロパティ 161
- SRV\_S\_LOGSIZE プロパティ 98, 161
- SRV\_S\_MAXLISTENERS プロパティ 96, 162
- SRV\_S\_NETBUFSIZE プロパティ 162
- SRV\_S\_NETTRACEFILE プロパティ 162
- SRV\_S\_NUMCONNECTIONS プロパティ 163
- SRV\_S\_NUMLISTENERS プロパティ 96, 163
- SRV\_S\_NUMMSGQUEUES プロパティ 163
- SRV\_S\_NUMMUTEXES プロパティ 163
- SRV\_S\_NUMREMBUF プロパティ 163
- SRV\_S\_NUMREMSITES プロパティ 163
- SRV\_S\_NUMTHREADS プロパティ 163
- SRV\_S\_NUMUSEREVENTS プロパティ 163
- SRV\_S\_PREEMPT プロパティ 129, 164
- SRV\_S\_REALLOCFUNC プロパティ 164
- SRV\_S\_REQUEST\_CAP プロパティ 164
- SRV\_S\_RESPONSE\_CAP プロパティ 164
- SRV\_S\_RETPARAMS プロパティ 164
- SRV\_S\_RETPARMS プロパティ 164
- SRV\_S\_SEC\_PRINCIPAL プロパティ 165, 207
- SRV\_S\_SERVERNAME プロパティ 165
- SRV\_S\_STACKSIZE プロパティ 166, 471
- SRV\_S\_TCP\_RCVBUF 380
- SRV\_S\_TCP\_SNDBUF プロパティ 380
- SRV\_S\_TDSVERSION プロパティ 166, 167, 183
- SRV\_S\_TIMESLICE プロパティ 166
- SRV\_S\_TRACEFLAG プロパティ 166, 167
- SRV\_S\_TRUNCATELOG プロパティ 166
- SRV\_S\_USERVLANG プロパティ 166
- SRV\_S\_USESRVLANG プロパティ 112, 119, 177
- SRV\_S\_VERSION プロパティ 166
- SRV\_S\_VIRTCLKRATE プロパティ 167
- SRV\_S\_VIRTIMER プロパティ 167
- SRV\_SECLABEL 値 172
- srv\_select (UNIX のみ) 427, 431
- srv\_send\_ctlinfo 431
- srv\_send\_data 433
- srv\_send\_text 228, 431, 440
- srv\_senddone 440, 445
- srv\_sendinfo 38, 445, 449
- srv\_sendpassthru 149, 449, 452
- srv\_sendstatus 452, 453
- SRV\_SERVER 構造体 340
- srv\_setcolutype 453, 454

- srv\_setcontrol 454, 457
- srv\_setloginfo 31, 457, 460
- SRV\_SETOPTION 値 139
- srv\_setpri 460, 462
  - マルチスレッド・プログラミング 132
- srv\_signal (UNIX のみ) 462
- SRV\_SITEHANDLER ログイン・タイプ 181
- srv\_sleep 127, 128, 464, 468
- srv\_spawn 468, 471
- SRV\_START イベント 101, 107
- SRV\_START ハンドラ 102
- SRV\_STOP イベント 101, 104, 107, 314, 317
  - SRV\_SERVER 構造体 340
  - 致命的なエラー 99
- SRV\_SUBCHANNEL ログイン・タイプ 181
- srv\_symbol 471, 472
- SRV\_T\_APPLNAME プロパティ 169
- SRV\_T\_BULKTYPE プロパティ 169, 177
- SRV\_T\_BYTEORDER プロパティ 169
- SRV\_T\_CHARTYPE プロパティ 178
- SRV\_T\_CHARYPE プロパティ 169
- SRV\_T\_CIPHER\_SUITE プロパティ 169
- SRV\_T\_CLIB プロパティ 169
- SRV\_T\_CLIBVERS プロパティ 169
- SRV\_T\_CLIENTLOGOUT プロパティ 169
- SRV\_T\_CONVERTSHORT プロパティ 169
- SRV\_T\_DUMPLOAD プロパティ 170
- SRV\_T\_ENDPOINT プロパティ 170
- SRV\_T\_EVENT プロパティ 170, 179
- SRV\_T\_EVENTDATA プロパティ 170
- SRV\_T\_FLTTYPE プロパティ 170, 180
- SRV\_T\_FULLPASSTHRU プロパティ 170
- SRV\_T\_GOTATTENTION プロパティ 21, 170
- SRV\_T\_HOSTNAME プロパティ 170
- SRV\_T\_HOSTPROCID プロパティ 170
- SRV\_T\_IODEAD プロパティ 171
- SRV\_T\_LISTENADDR プロパティ 96, 171
- SRV\_T\_LOCALE プロパティ 171
- SRV\_T\_LOCALID プロパティ 96, 171
- SRV\_T\_LOGINTYPE プロパティ 171, 180
- SRV\_T\_MACHINE プロパティ 171
- SRV\_T\_MIGRATE\_STATE 171
- SRV\_T\_MIGRATED 49
- SRV\_T\_MIGRATED プロパティ 171
- SRV\_T\_NEGLOGIN プロパティ 172
- SRV\_T\_NOTIFYCHARSET プロパティ 172
- SRV\_T\_NOTIFYDB プロパティ 172
- SRV\_T\_NOTIFYLANG プロパティ 172
- SRV\_T\_NUMRMPWDS プロパティ 172
- SRV\_T\_PACKETSIZE プロパティ 172
- SRV\_T\_PASSTHRU プロパティ 173
- SRV\_T\_PRIORITY プロパティ 173
- SRV\_T\_PWD プロパティ 173
- SRV\_T\_REMOTEADDR プロパティ 96, 173
- SRV\_T\_RETPARAMS プロパティ 173
- SRV\_T\_RMTPWD 構造体 182
- SRV\_T\_RMTPWDS プロパティ 173, 182
- SRV\_T\_RMTSERVER プロパティ 173
- SRV\_T\_ROWSENT プロパティ 174
- SRV\_T\_SEC\_CHANBIND プロパティ 174
- SRV\_T\_SEC\_CONFIDENTIALITY
  - プロパティ 174
- SRV\_T\_SEC\_CREDTIMEOUT プロパティ 174
- SRV\_T\_SEC\_DATAORIGIN プロパティ 174
- SRV\_T\_SEC\_DELEGATION プロパティ 174
- SRV\_T\_SEC\_DELEGCRED プロパティ 175
- SRV\_T\_SEC\_DETECTREPLAY プロパティ 175
- SRV\_T\_SEC\_DETECTSEQ プロパティ 175
- SRV\_T\_SEC\_INTEGRITY プロパティ 175
- SRV\_T\_SEC\_MECHANISM プロパティ 175
- SRV\_T\_SEC\_MUTUALAUTH プロパティ 175
- SRV\_T\_SEC\_NETWORKAUTH プロパティ 175
- SRV\_T\_SEC\_SESSTIMEOUT プロパティ 176
- SRV\_T\_SESSIONID 49
- SRV\_T\_SESSIONID プロパティ 176
- SRV\_T\_SPID プロパティ 176
- SRV\_T\_SSL\_VERSION プロパティ 176
- SRV\_T\_STACKLEFT プロパティ 176
- SRV\_T\_TDSVERSION プロパティ 176
- SRV\_T\_TYPE プロパティ 176, 184
- SRV\_T\_USER プロパティ 177
- SRV\_T\_USERDATA プロパティ 177
- SRV\_T\_USESRVLANG プロパティ 112, 119, 177
- SRV\_T\_USTATE プロパティ 177
- srv\_tabcolname 476, 478
  - ブラウザ・モードの結果を返すための呼び出し 24
- srv\_tabname 478, 480
  - ブラウザ・モードの結果を返すための呼び出し 24
- SRV\_TCLIENT スレッド・タイプ 184
- SRV\_TDS\_ 値 167, 184

srv\_termproc 288, 480, 481  
 srv\_text\_info 61, 226, 482, 485  
 srv\_thread\_props 158, 168, 485, 486  
 srv\_thread\_props プロパティ 21  
 SRV\_TIMESLICE 設定パラメータ 133  
 SRV\_TLISTENER スレッド・タイプ 96, 184  
 SRV\_TR\_ATTEN 値 168  
 SRV\_TR\_DEFQUEUE 値 168  
 SRV\_TR\_EVENT 値 168  
 SRV\_TR\_MSGQ 値 168  
 SRV\_TR\_NETDRIVER 値 168  
 SRV\_TR\_NETREQ 値 168  
 SRV\_TR\_NETWAKE 値 168  
 SRV\_TR\_TSDATA 値 168  
 SRV\_TR\_TSDHDR 値 168  
 SRV\_TSERVICE スレッド・タイプ 184  
 SRV\_TSITE スレッド・タイプ 184  
 SRV\_TSUBPROC スレッド・タイプ 184  
 srv\_ucwakeup 20, 492  
 SRV\_UNITEXTLOAD 値 178  
 srv\_unlockmutex 495  
 SRV\_URGDISCONNECT  
   イベント 101, 108, 314, 317  
 srv\_version 495, 497  
 SRV\_VIRTCLKRATE 設定パラメータ 133  
 SRV\_VIRTTIMER 設定パラメータ 133  
 srv\_wakeup 20, 127, 492, 497, 500  
 srv\_xferdata 152, 156, 268, 500, 503  
 srv\_yield 127, 129, 503, 504  
 SSL  
   SDC 204  
   証明書 204

## T

### TCL

Net-Lib ドライバ要求 168  
 ウェイクアップ要求 168

### TDS

srv\_props による初期バージョン値の指定  
   166  
 srv\_thread\_props によるクライアントの thread  
   ¥xd5 s バージョンの取得と設定 176  
 定義 3

パススルー・モード 111  
 プロトコル・レベル 135  
 TDS バージョン 167  
   機能 38  
   ネゴシエーション 183  
   有効な値 167  
 TDS パケット  
   パススルー・モード 146  
   ヘッダ情報 168  
 Tabular Data Stream プロトコル TDS を参照 3  
 text および image データ  
   クライアントからの取得 226, 228  
   クライアントへの送信 226  
 text および image データ型 241  
 text データ型 27, 29, 61, 225, 239, 241  
   srv\_get\_text 320, 321  
   テキスト・タイムスタンプ 225  
   テキスト・ポインタ 225  
   転送 178  
 text と image 225, 228

## U

unitext データ型  
   srv\_get\_data 320  
 use db コマンド 172

## W

writetext ストリーム 226

## X

XML のデータ型 234

## あ

アテンション 170  
   SRV\_ATTENTION イベント・ハンドラ 20  
   srv\_thread\_props によるチェック 21

- コーディングの考慮事項 21
- 割り込みレベル 20
- アドホック・ネゴシエーション 137
- アプリケーションで定義されたログイン・ハンド  
シェイク 136, 172
- アプリケーション名 338
- 暗号化 174
  - key 136
  - パスワード 172
- 暗号化シグニチャ 175

## い

- 委任クレデンシャル 175
- イベント 100, 109
  - srv\_event 313
  - アテンション 20
  - カーソル 68, 77
  - 処理 9
  - 切断 22
  - 通知 27
  - 定義 100
  - 動的SQL 91
  - 標準のリスト 102, 108
  - プログラマ定義 108
  - メッセージ 86
- イベント・キュー 126
- イベント駆動型スレッド 123
- イベント・ハンドラ
  - srv\_capability 37
  - srv\_handle 335
  - カスタム・ハンドラのコーディング 102
  - 定義 101
  - デフォルト 101
  - デフォルトとカスタムの比較 101
  - メッセージ 88
  - 割り込みレベル 20
- インストール
  - イベント・ハンドラ 335
  - エラー・ハンドラ 97

## う

- 埋め込み 59

## え

- エラー 38, 64, 95, 100
  - CS-Library 64
  - 拡張データ 40
  - 各国言語メッセージ 112, 115
  - カラム・レベルの情報 40
  - 「クライアント・コマンド・エラー」参照 38
  - 重大度 98
  - タイプ 97
  - 番号 99
- エラーの重大度 98
- エラー・ハンドラ 64, 97, 161
  - インストール 9, 18
  - 環境変数の変更 312
- エラー・メッセージ 38
  - クライアントへの送信 38
  - トラブルシューティング 527

## お

- 応答 164
- 応答機能の表 29, 30
- オープン、カーソル・コマンド 69
- オプション 138, 143
  - 設定と取得 138, 139
  - 説明 139, 143
  - デフォルト値 139, 143
- オペレーティング・システム・エラー 99

## か

- カーソル 27, 66, 83
  - CS\_DATAFMT 構造体 60
  - ID 69
  - SRV\_CURDESC 構造体 70, 82
  - SRV\_CURSOR イベント・ハンドラ 77
  - srv\_cursor\_props 288, 291



- カーソル・コマンドのタイプ 68
  - カーソル要求の処理 77, 82
  - キー・データ 82
  - 更新 71, 82
  - 更新カラム 82
  - 更新テキスト 77
  - サーバ・オプション 140
  - 使用の利点 67
  - 定義 67
  - フェッチ・タイプ 72
  - ローのフェッチ 26
  - カーソル・ハンドラ SRV\_CURSOR イベント・ハンドラを参照 241
  - カーソル・コマンド 152
  - 解析解決ツリー 142
  - 解放、C ルーチン 319
  - 解放、メモリ 161, 318
  - 隠しカラム
    - CS\_DATAFMT 構造体 60
  - 拡張エラー・データ 39, 41
    - クライアントへの送信 40
    - 定義 39
  - 仮想タイマ 167
  - 各国言語 135, 159, 166, 177
    - 再ネゴシエーション 137
    - 変更の通知 172
  - 各国言語と文字セット 111
    - 変更 117
  - 可変長 binary データ型 26
    - long 26
  - カラム
    - オリジナル名 56
  - 環境の変化 310
  - 環境変数 312
- き**
- キー 82
  - 機能 136, 276, 284
    - 1つずつのネゴシエーション 35
    - 10.0 以前のクライアントとのネゴシエーション 38
    - CS\_CAP\_TYPE 構造体 36
    - srv\_props によるデフォルト値の変更 30
    - TDS バージョン 32, 38
    - 応答機能の表 29, 30
    - 機能マクロ 36, 37
    - クライアント接続 278
    - 使用 25
    - デフォルト 30
    - デフォルト値のリスト 31, 35
    - 透過的ネゴシエーション 30
    - 特定の検索 37
    - ネゴシエーション 25
    - ビットマスク 36
    - マクロ 36
    - 明示的ネゴシエーション 35
    - 要求機能の表 25, 27
  - 接続属性機能を参照 25
  - 接続ハンドラ SRV\_CONNECT イベント・ハンドラを参照 241
  - 接続解除ハンドラ SRV\_DISCONNECT イベント・ハンドラを参照 241
  - 共通名の検証
    - SDC 環境 204
  - 共通ライブラリ 63
  - 共有ディスク・クラスタ環境
    - 証明書 204
  - 切換
    - スレッド・コンテキスト 295
- く**
- クエリ
    - 構文 142
    - 情報 141
    - 処理動作 137
  - クライアント
    - クライアントのタイプ 2
    - 定義 2
    - ログイン情報 324
  - クライアント・コマンド・エラー
    - CS\_SERVERMSG 構造体 38
    - srv\_sendinfo による送信 38, 39
  - クライアント/サーバ

- アーキテクチャ 1, 2
- クライアント/サーバ動作
  - の再ネゴシエーション 137
- クライアント・スレッド 126, 184
- クライアントの認証 136
- クライアント要求 136, 164
- クライアント・ログアウト 169
- クライアント・ログイン要求 181
- クレデンシャル 196
  - 委任 175
  - タイムアウト 174
- クローズ、カーソル・コマンド 69
- クロック・レート 167

## け

- 計算ロー 250, 259
  - クライアントに送信 261
  - 集合 255
- ゲートウェイ・アプリケーション 109, 111, 113, 116, 139, 146
  - srv\_getloginfo 325
  - アテンション 21
  - 個別のセキュリティ・セッション 213
  - ダイレクト・セキュリティ・セッション 213, 219
- 結果
  - 概要 16
  - 処理 16
  - 返信の順序 17
- 言語
  - コマンド 152
  - データ・ストリーム 157
  - 要求 105
  - 呼び出し 120
- 言語ハンドラ SRV\_LANGUAGE
  - イベント・ハンドラを参照 241

## こ

- 更新 141
  - カーソル 67, 69, 71, 82
- コール・スタック、スレッド 292

- コールバック
  - インストール 272
  - セキュリティ・セッション 219, 222, 225
  - タイムスライス 166
- 状態遷移ハンドラ「コールバック」参照 272
- コールバック・ハンドラ
  - エラー 64
  - スレッドのためのインストール 132
  - レジスタード・プロシージャ 190
- コンテキスト切り替え 127
- コンテキスト・プロパティ
  - cs\_config 158
  - ct\_config 158
  - srv\_props 159
  - 定義 158
- コンテキスト構造体「CS\_CONTEXT 構造体」
  - 参照 241

## さ

- サード・パーティのセキュリティ 196
- サーバ
  - サーバのタイプ 2
  - サーバ・エラー・メッセージ 64
  - トラブルシューティング 527
  - サーバ・プロパティ
    - 定義 159
  - サーバ名
    - srv\_props による指定 165
  - サービス・スレッド 109, 123, 126, 184
- 最大ロー 142
- サイト・ハンドラ 123, 184
  - srv\_props による数の設定 163
  - サブチャンネル・ログイン 181
  - ログイン要求 181
- 削除 141
- 削除、カーソル・コマンド 69
- 算術例外 140

## し

- 識別子 142
- シグナル (UNIX) 317

システム・レジスタード・プロシージャ  
 定義 189  
 マッピング、Server-Library ルーチン 189  
 実行キュー 129  
 集合  
 計算ロー 255  
 週、初日 140  
 受信アドレス 89  
 準備  
 文 308  
 準備文 90  
 ジョイン 141  
 情報エラー 99  
 情報、カーソル・コマンド 69  
 証明書  
 SSL 204  
 初期化  
 手順のまとめ 159  
 プロパティの設定 159  
 初日、週 140

## す

スタック・サイズ  
 スレッド 166  
 スタック領域  
`srv_thread_props` による設定 176  
 ステータス値  
 クライアントへの返信 17  
 ステータスの移行  
`srv_callback` 133  
 スリープ中のスレッド 129  
 スレッド  
 ID 176  
 現在のステータス 177  
 コール・スタック 292  
 コンテキストの切替 295  
 使用可能数の設定、`srv_props` 163  
 スタック・サイズ 166  
 ステータスの移行 272  
 スレッド・プロパティ・テーブル 169, 177  
 タイプ 123, 184  
 タイプ、`srv_thread_props` による

`client` ¥`xd5 s` の取得 176  
 通信 131  
 定義 123  
 非クライアント 286  
 プリエンプティブ 124  
 プロパティ 168, 182  
 マルチスレッド・プログラミングを参照 123  
 メッセージ 122  
 ログイン・レコード 180  
 スレッドのスケジューリング 127, 130

## せ

整合性サービス 175  
 セキュア接続 136  
 確立のためのクライアントとのネゴシエーション 136  
 セキュリティ・サービス 196, 225  
 スレッド・プロパティ 174  
 セキュリティ・セッション  
 ゲートウェイ・アプリケーション 213  
 タイムアウト 176  
 単純な Open Server アプリケーション 213  
 セキュリティ・セッション  
 ・コールバック 219, 222, 225  
 セキュリティ・メカニズム 196  
`interfaces` ファイル 211  
 ローカル名 175, 207  
 セキュリティ・ラベル 136, 140, 172  
 セキュリティ・レベル 136  
 ネゴシエーション 136  
 接続マイグレーション 41  
 切断  
 処理 22  
 説明  
 カラムおよびパラメータ 302  
 宣言、カーソル・コマンド 69

## そ

相互認証 175  
 送信

クライアントへのメッセージ 87  
 ロー・データ 152  
 挿入 141  
 ソート順 111, 119, 159  
 情報を返す 119

## た

タイプ 229, 241  
 タイム・スライス・コールバック 166  
 ダンプ/ロード 170

## ち

遅延イベント  
 キュー・サイズ 161  
 致命的なエラー 99  
 チャネル・バインド 174, 196  
 チャレンジ/応答 172  
 仲介アプリケーション 109  
 中断スレッド 129

## て

ディレクトリ・サービス 88, 90  
 ディレクトリ・サービス・プロバイダ 161  
 ディレクトリ・ドライバ 88  
 データ  
 オリジン 174  
 記述、バインド、転送 155  
 機密保持 174  
 整合性 175  
 データ型 231  
 CS\_BIGDATETIME 234  
 CS\_BIGTIME 234  
 CS\_DATE 234  
 CS\_DATETIME 234  
 CS\_DATETIME4 234  
 CS\_TIME 234  
 応答機能 30  
 操作するルーチン 231  
 タイプを参照 228

データ型のサマリテーブル 229, 231  
 データの記述 155  
 データの転送 156  
 デバッグ 275, 295  
 デフォルト・イベント・ハンドラ 101

## と

透過的ネゴシエーション 30, 135  
 機能 25  
 同時実行性 127, 132  
 動的 SQL 27, 90, 95  
 CS\_DATAFMT 構造体 60  
 srv\_dynamic 306  
 SRV\_DYNAMIC イベント・ハンドラ 91  
 srv\_dynamic ルーチン 91  
 カーソル 67  
 クライアント動的 SQL コマンドへの応答 91  
 コマンド 152  
 使用 90  
 使用の利点 90  
 動的 SQL ハンドラ SRV\_DYNAMIC  
 イベント・ハンドラを参照 241  
 動的なリスナ 95  
 起動 96  
 スレッド・タイプ 96  
 スレッド・プロパティ 96  
 設定 95  
 プロパティ 96  
 登録  
 ディレクトリ 89, 161  
 トランザクション独立性レベル 141  
 トレース 166, 168  
 トレース・フラグ  
 Open Server トレース・フラグのまとめ 168

## な

内部 I/O 統計 142

## に

二重引用符、識別子 142  
日時形式 111

## ね

ネーミング・サービス 88, 90  
ネゴシエーション  
  SRV\_CONNECT イベント・ハンドラ 135  
  TDS プロトコル・レベル 147, 151  
  オプション・コマンドまたは言語コマンド 137  
  機能 25  
  透過的 30  
ネゴシエートされた動作 135, 137  
ネゴシエートされたパケット・サイズ 172  
ネゴシエートされたログイン  
  srv\_thread\_props によるクライアント要求の取得 172  
  コマンド 152  
ネットワーク I/O バッファ  
  srv\_props によるサイズの設定 162  
ネットワーク接続  
  srv\_props による数の設定 163  
ネットワーク認証 175

## の

ノーティフィケーション  
  レジスタード・プロシージャ 186  
ノーティフィケーション・プロシージャ 186

## は

バージョンの文字列 166  
バイト  
  コピー 269  
バイト順 135  
  srv\_thread\_props によるスキームの取得 169  
バインド  
  変数 263

バインド、データ 155  
パケット・サイズ 172  
パススルー・モード 111, 143, 149, 173  
  TDS レベルのネゴシエーション 147, 151  
  ゲートウェイ 143  
  ダイレクト・セキュリティ・セッションでのゲートウェイ 213, 219  
  ルーチン 148  
パスワード  
  srv\_thread\_props による client¥xd5 s の取得 173  
パラメータ  
  RPC 193  
  クライアントからの取得 153  
  リターン・パラメータ 17  
パラメータ・データ 152  
パラメータ・データの取得 152  
パラメータとロー・データの処理 151  
バルク  
  コピー要求 103  
  挿入 170  
  データ転送 169, 177  
バンド外アテンション 26  
バンド内アテンション 26

## ひ

非クライアント・イベント 101  
非クライアント・スレッド 286  
日付  
  要素の順番 141  
ビットマスク  
  CS\_BROWSEDESC 構造体 56  
  CS\_DATAFMT status 値 60  
  CS\_KEY 82  
  機能 36  
非同期イベント 315  
非標準 SQL 141  
非プリエンプティブ・スケジューリング  
  srv\_props による指定 164  
  定義 128  
コルーチン・スケジューリング非プリエンプティブ・スケジューリングを参照 128

標準イベント 101  
非連鎖トランザクション 140

## ふ

ファイル記述子  
終了ポイント 170  
フェッチ・タイプ 72  
浮動小数点表現 135, 180  
ブラウズ・モード 55  
CS\_BROWSEDESC 構造体 24  
クライアントにブラウズ  
・モードの結果を返す 24  
サポートする手順 24  
プラットフォーム依存のサービス 276  
プラットフォーム機能 278, 282  
プリエンプティブ・スレッド  
・スケジューリング 128, 129  
srv\_props による指定 163, 164  
定義 128  
プリエンプティブ・スレッド 124  
スケジューリング 277  
プリンシパル 207  
プログラマ定義イベント 101, 108  
プロセス ID  
クライアント 170  
プロトコル機能 278, 282  
プロバイダ、ディレクトリ・サービス 88  
分散サービス・プロバイダ 196

## ほ

ホスト・マシン、クライアント 170

## ま

マクロ  
機能 36  
マルチスレッド・プログラミング 123, 134  
srv\_setpri 132  
概要 18  
コールバック・ハンドラ 132, 133

スレッド・スケジューリング 127, 130  
スレッドのタイプ 123, 127  
スレッドの定義 123  
ツールと手法 130, 133  
プログラミングに関する注意事項 133, 134  
ミューテックス 130  
メッセージ・キュー 131

## み

ミューテックス  
srv\_getobjname 331  
srv\_props による数の設定 163  
オブジェクト ID 329  
削除 300  
作成 284  
定義 130

## め

明示的ネゴシエーション 25, 135  
メッセージ 27, 152  
ID 86  
Open Server でのタイプ 121  
イベント・ハンドラ 88  
エラー 38, 527  
クライアントからの取得 86  
重大度 65  
受信 87  
使用可能な数 162  
定義 86  
データ・パラメータ 268  
テキスト長 39  
番号 65  
連続化 39, 65  
メッセージ・ハンドラ SRV\_MSG  
イベント・ハンドラを参照 241  
メッセージ・イベント 86  
メッセージ・キュー  
srv\_getmsgq 326  
srv\_getobjname 331

srv\_props による数の設定 163  
 アクティビティ 168  
 オブジェクト ID 329  
 削除 298  
 作成 282  
 定義 131  
 データ・ストリーム・メッセージ  
   「メッセージ」参照 86  
 メッセージの受信 87  
 メッセージ・リプレイ 175  
 メッセージ・リプレイの検出 175  
 メモリ  
   srv\_free 318  
   ゼロに設定 271  
   バイトの移動 269  
   ルーチン解放、srv\_props による指定 161  
   ルーチン再割り付け、srv\_props  
     による指定 164  
   割り付け 160, 164, 248

## も

文字セット 111, 135, 159  
   クライアントからの変更要求の処理 117  
   再ネゴシエーション 137  
   情報を返す 119  
   変更 118  
   変更の通知 172  
 文字データ表現 169, 178

## ゆ

有効バイト 169  
 ユーザ・イベント  
   定義 295  
   番号 163  
 ユーザ権限 140  
 ユーザ定義イベント 315, 317  
 ユーザ名  
   srv\_thread\_props による  
     client $\forall$ xd5 s の取得 177  
 優先順位レベル 129, 173

## よ

要求  
   動的 SQL 91  
 要求機能の表 25, 27

## ら

ラージ・オブジェクト (LOB)  
   ロケータのデータ型 239

## り

リアルタイム・データ 130  
 リターン・パラメータ 152, 154  
   SRV\_LANGUAGE  
     イベント・ハンドラでの処理 156  
   言語データ・ストリーム 157  
   処理 154  
 リモート・サーバ 110  
   srv\_thread\_props による名前の取得 173  
   セキュリティ・セッション 196  
   パスワード 173, 182  
 リモート・パスワード 172  
   srv\_thread\_props による取得 172  
 リモート・パスワード、  
   srv\_thread\_props による取得 172  
 リモート・プロシージャ・コール 27, 106, 152,  
   192, 193  
   CS\_DATAFMT 構造体 60  
   処理 192  
   定義 192  
 RPC リモート・プロシージャ  
   ・コールを参照 192

## れ

レジスタード・プロシージャ  
   コールバック・ハンドラの使用 190

- 実行 188
- 定義 16, 185
- 登録の手順 187
- リストの管理 188
- 利点 185
- リモート・プロシージャ・コールとの比較 185
- 連鎖トランザクション 140
- 連続化 65
  - メッセージ 39

## ろ

- ロー
  - 影響を受けた 141
  - 最大 142
  - 処理 23, 154
- ローカライズされたクライアント 111, 113
- ローカライゼーション 111, 120, 159
  - CS\_CONTEXT 構造体 116, 117
  - Open Server アプリケーション 112, 114
  - sp\_serverinfo 119
  - CS\_LOCALE 構造体 113, 114
  - 関連プロパティ 119
  - クライアントヘローカラ
    - イゼーション情報を返す 119
  - ローカライズされたクライアントのサポート 113, 117
  - ローカライズされた接続の作成 116
- 国際化サポートローカライゼーションを参照 111
- ローカル言語 135
- ロー・データ 152
- ローのフェッチ 26, 69
- ローの返送 154
- ログアウト、クライアント 169
- ログイン・ネゴシエーション 135
- ログイン要求 181
- ログ・ファイル 98, 115, 131
  - srv\_props によるサイズの設定 161
  - srv\_props による指定 161
  - 起動時のトランケート 166
  - 最大サイズ 161
  - 名前 161
- ロック 130

## わ

- 割り込み 20, 103, 134, 161
- 割り込みレベル
  - Server-Library 呼び出し 20
  - アテンション 20
- 割り付け
  - メモリ 248
- 割り付け解除、カーソル・コマンド 69
- 割り付け、メモリ 160