Sybase, Inc.
One Sybase Drive
Dublin, CA 94568
www.sybase.com

Sybase Avaki EII API Guide

Release 7.0

August 24, 2006

DOCUMENT ID: DC00576-01-0700-01

LAST REVISED: August 2006

Copyright © 2002 – 2006 Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase, SYBASE (logo), ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Advantage Database Server, Afaria, Answers Anywhere, Applied Meta, Applied Metacomputing, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Translator, APT-Library, ASEP, Avaki, Avaki (Arrow Design), Avaki Data Grid, AvantGo, Backup Server, BayCam, Beyond Connected, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional Logo, ClearConnect, Client-Library, Client Services, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DataWindow .NET, DB-Library, dbQueue, Dejima, Dejima Direct, Developers Workbench, DirectConnect Anywhere, DirectConnect, Distribution Director, Dynamic Mobility Model, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, EII Plus, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise Portal (logo), Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, ExtendedAssist, Extended Systems, ExtendedView, Financial Fusion, Financial Fusion (and design), Financial Fusion Server, Formula One, Fusion Powered e-Finance, Fusion Powered Financial Destinations, Fusion Powered STP, Gateway Manager, GeoPoint, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, Intelligent Self-Care, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Legion, Logical Memory Manager, lrLite, M2M Anywhere, Mach Desktop, Mail Anywhere Studio, Mainframe Connect, Maintenance Express, Manage Anywhere Studio, MAP, M-Business Anywhere, M-Business Channel, M-Business Network, M-Business Suite, MDI Access Server, MDI Database Gateway, media.splash, Message Anywhere Server, MetaWorks, MethodSet, mFolio, Mirror Activator, ML Query, MobiCATS, MobileQ, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASiS, OASiS logo, ObjectConnect, ObjectCycle, OmniConnect, OmniQ, OmniSQL Access Module, OmniSQL Toolkit, OneBridge, Open Biz, Open Business Interchange, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Pharma Anywhere, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power++, Power Through Knowledge, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Pylon, Pylon Anywhere, Pylon Application Server, Pylon Conduit, Pylon PIM Server, Pylon Pro, QAnywhere, Rapport, Relational Beans, RemoteWare, RepConnector, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RFID Anywhere, RW-DisplayLib, RW-Library, SAFE, SAFE/PRO, Sales Anywhere, Search Anywhere, SDF, Search Anywhere, Secure SQL Server, Secure SQL Toolset, Security Guardian, ShareLink, ShareSpool, SKILS, smart.partners, smart.parts, smart.script, SOA Anywhere Trademark,SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, S.W.I.F.T. Message Format Libraries, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase Virtual Server Architecture, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SybFlex, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TotalFix, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viafone, Viewer, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, XcelleNet, XP Server, XTNDAccess and XTNDConnect are trademarks of Sybase, Inc. or its subsidiaries. 07/06

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

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.

Set in Arial, `Courier New`, and Times New Roman. Stanley Morison, the creator of Times New Roman, said of it: “By the vice of Mammon and the misery of the machine, it is bigoted and narrow, mean and puritan.”

Credits
This product includes software developed by the Apache Software Foundation (http://www. apache.org). This product includes Hypersonic SQL and ANTLR. This product includes code licenses from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/. Contains IBM® 64-bit Runtime Environment for AIX™, Java™ 2 Technology Edition Version 1.4 Modules © Copyright IBM Corporation 1999, 2000 All Rights Reserved. Contains the SAXON XSLT Processor from Michael Kay, which is available at http://saxon.sourceforge.net. This product includes software developed by the Proxool Project (http://proxool.sourceforge.net).

Sybase Avaki EII API Guide
Written by Meryl R. Cohen, Beth Thoenen, and Cheryl Magadieu

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

Table of contents

Preface

This *Sybase Avaki EII API Guide* explains how to use Sybase Avaki EII development tools. The guide is intended for programmers who want to use Avaki web services or access or manipulate Avaki database data programmatically.

Users are expected to have:

- Basic Java and JDBC programming knowledge.
- Knowledge of the components, structure, and features of Avaki data grids. See the *Sybase Avaki EII Overture* and the *Sybase Avaki EII Administration Guide* for details.

Note This book and the product's user interfaces refer to Sybase Avaki EII software as *Avaki* or *Avaki Data Grid*.

Organization

This book is organized as follows:

Chapter 1 Using the web services API	Tells you how to use the web services application programming interface to programmatically invoke data services, database operations, ad-hoc queries, and data catalog operations.
Chapter 2 Web services API reference	Describes the complex types, SOAP operations, data catalog operations, data service operations, and database operations of the web services API.
Chapter 3 Using JDBC drivers	Explains how to access and use the Avaki JDBC driver in Java applications.
Glossary	Defines terms used in this guide

Related documentation

These manuals make up the Avaki documentation set:

- *Sybase Avaki EII Overture*
- *Sybase Avaki EII Administration Guide* (includes installation instructions)
- *Data Integration with Sybase Avaki Studio*
- *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*
- *Sybase Avaki EII API Guide*
- *Sybase Avaki EII Command Reference*

The manuals are included, in PDF format, on the CD with the Avaki software. They are stored in the docs subdirectory of the Avaki installation directory.

To access the manuals via Avaki's web user interface, log in to your Avaki domain and click the **Help** link at the top right corner of any page of the web UI.

Conventions

This table describes conventions this book uses in examples of user input and system output.

Convention	Description	Example
`$` or `C:>`	The command prompt	`$` or `C:>`
`< >`	A placeholder; replace the content inside the brackets with an option or value	`$` **`avaki ls`** `<grid-path>`
`screen font`	Text that appears on the screen	`sample text`
`bold screen font`	User input—commands that you enter	`$` **`avaki ls`**

How to contact Avaki support at Sybase, Inc.

For general information about Sybase technical support, see the *Customer Service Reference Guide* at

http://www.sybase.com/support/aboutsupport/guide/csrg

Please contact us with any questions or difficulties you encounter.

By telephone

In North America, call toll free: 1-800-8SYBASE

Outside North America, follow the link below to see a list of Sybase offices and phone numbers around the world.

http://www.sybase.com/contactus/support

On the web

If you are a designated contact for a technical support plan, you can log and track cases on the web using the Case Express application. At www.sybase.com, mouse over the **Support and Services** tab and select **Case Management** from the dropdown list. Use the email address and password for your mysybase account to log in.

Chapter 1

Using the web services API

Avaki web services are a mechanism to access data in a data grid and use it in applications. Web services architectures allow different applications to interact in a language-independent way. Using web services promotes ease of application programming and maintenance, as well as portability of the code.

Web services can be implemented in a variety of ways. Avaki supports the SOAP (simple object access protocol) over HTTP protocol. The Avaki WS API (web services API) provides server-side support and the information you need to write a client application.

Specifically, Avaki is implementation-compliant with SOAP version 1.1 and WSDL version 2.0. For the formal SOAP definition, see `http://www.w3.org/TR/SOAP`.

Chapter overview: This chapter describes general features of the Avaki WS API and provides information you need before beginning to implement your SOAP client. We cover the following topics:

- For a brief overview of Avaki's WS API, see "Web services overview" on page 3.
- For information on aspects of web service client development with the WS API, see "Web service client considerations" on page 5.
- For examples of web service clients, see "Web services client examples" on page 10.

Chapter 2, "Web services API reference", contains a reference for the WS API.

Audience: We expect readers of this chapter to have experience with SOAP, to have knowledge of the programming language in which their web services client will be written, and to know how to use their SOAP client development environment (see "Choose a web service development framework" on page 5). If you do not have this background but would like to learn more about web services, here are some resources.

- This web page is useful as a starting point for general web service concepts, even if you are not using .NET.
  Understanding Distributed Technologies
  `http://msdn.microsoft.com/webservices/understanding/default.aspx`

- This web page provides a good overview of SOAP.
  Understanding SOAP
  `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/understandsoap.asp`

- The Soap Learning Guide is a web page with links to many SOAP resources.
  `http://searchwebservices.techtarget.com/originalContent/1,289142,sid26_gci913069,00.html`

- This web page is primarily an introduction to the Axis Java web services development framework. Also provides some information on web services in general.
  Axis User's Guide
  `http://ws.apache.org/axis/java/user-guide.html`

Avaki configuration: Your Avaki domain must already be deployed and provisioned with any data and services that your SOAP web service client will access before you can write and run your client. Access controls for data or services accessed by the client must also be set appropriately. These tasks are described in the following manuals:

- The *Sybase Avaki EII Administration Guide* describes how to set up an Avaki domain, including how to create and import user accounts and groups, how to provision the domain with files shared from local file systems, and how to set access controls for objects in the domain.

- The *Sybase Avaki EII Provisioning and Advanced Data Integration Guide* describes how to use the web UI to provision an Avaki domain with database operations, data services, and SQL views. You can also use Avaki Studio to create database operations and data services—see *Data Integration with Sybase Avaki Studio* for instructions.

Web services overview

The Avaki web services and their WSDL documents

The Avaki WS API facilitates data access using the SOAP protocol. The WS API is an interface for accessing data within a data grid to use in third-party applications. It is not a method for adding data to the grid or performing any administrative grid functions. The operations that comprise the interface may be invoked in order to access grid data such as the results of a database operation, the contents of a grid directory, the results of a search, etc.

The WS API consists of four services, each of which is defined in a WSDL (web services description language) document provided by Avaki. A WSDL document is a formal statement of the contract between a SOAP client and a SOAP server. It is a platform- and language-independent XML document that describes the format of the SOAP requests expected by the server and the SOAP responses it generates. In addition, the WSDL specifies the name of the web service and the grid server and port where the service can be reached.

These are the Avaki web services and their WSDLs:

Web service	WSDL	Description
Avaki API, rpc/encoded version	AvakiAPIRpcEnc.wsdl	Defines the rpc/encoded versions of SOAP operations for Avaki web services.
Avaki API, document/literal version	AvakiAPIDocLit.wsdl	Defines the document/literal versions of the same SOAP operations for Avaki web services. This version is recommended for .NET clients.
Avaki API with MIME attachments, rpc/encoded version	AvakiAPIWithMIMERpcEnc.wsdl	Defines the rpc/encoded versions of the same SOAP operations for Avaki web services as well as special methods that return results as MIME attachments.
Avaki API with MIME attachments, document/literal version	AvakiAPIWithMIMEDocLit.wsdl	Defines the document/literal versions of the same SOAP operations for Avaki web services as well as special methods that return results as MIME attachments.

Note Along with the WSDL files, Avaki provides AvakiAPI.disco, which is a discovery file that .NET web service clients can use to discover WSDLs. A .NET client could browse through the discovery file to locate the WSDLs for Avaki services. Alternatively, the .NET client may bypass the discovery file altogether and use the WSDLs directly.

The web services listed in the previous table are nearly identical functionally. The differences between them can be summarized as follows:

- All four web services contain all operations provided by Avaki as part of the interface for accessing a grid.
- The Avaki API document/literal version and the rpc/encoded version are identical except that the style/usage for the operations is different.
- The Avaki API with MIME attachments versions are perfect supersets of the corresponding basic versions. In addition, the versions with MIME attachments provide operations that permit accessing the bulk results from certain accesses as MIME attachments.

Each web service is composed of many SOAP operations. The SOAP operations in each service of the WS API mirror their web UI and CLI counterparts. For details about the web UI, refer to the *Sybase Avaki EII Overture*, the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*, and the *Sybase Avaki EII Administration Guide*. For details on the CLI commands, refer to the *Sybase Avaki EII Command Reference*. For details on the WS API operations provided in each of these categories, see Chapter 2, "Web services API reference".

Web service client considerations

Before you begin developing your web service client, there are several topics to consider and choices to make.

Choose a web service development framework

While it is possible to write SOAP clients from scratch, most programmers use a development framework to help with the task. Tools in such frameworks abstract the details of constructing a SOAP request and parsing the SOAP response when making a web service call. Development frameworks are language-specific and may generate a code framework with stubs based on the specifications in the WSDL document. Choose a web service development framework that meets your particular language and development needs. There are many options available from third-party vendors, including the following, which have been tested against the Avaki WS API:

- **Java:** Apache Axis 1.3
- **Perl:** SOAP::Lite
- **Microsoft Visual Studio 2003**: VB .NET

All web service development frameworks generate their code by parsing the WSDL document that defines the SOAP interface for the service request and data returned.

Choose the appropriate WSDL document

Before developing your client, you must choose which web service to use. In effect, you choose the service by deciding which WSDL (from those listed in the table on page 3) to use for your client development since the WSDL defines the service. If you prefer document/literal web services, choose one of the WSDLs that supports such operations. If you require results returned as MIME attachments, choose a WSDL that contains operations that do return attachments. Typically, .NET clients will choose the AvakiAPIDocLit.wsdl WSDL file because .NET does not recognize MIME attachments and supports document/literal better. If you choose rpc/encoded services, beware that some operations (such as fileRead) return bulk data as base-64 encoded data that must be decoded on the client side. Those very operations in the document/literal versions return each byte enclosed in its own tag, thus reducing performance. Use the versions of the same operations that return strings or MIME attachments (fileReadString and fileReadAttach, for example).

Choose a grid server

All Avaki web services are accessible from any grid server. You might choose a particular grid server to balance the load across your deployment or because that server is closer to the application calling it. If you route your SOAP requests to a local grid server where caching is set up locally, the data grid's internal communications will be more efficient across a wide area and you will take better advantage of caching capability. The grid server that the web service will use is specified in the service URL in the WSDL. The section "Edit the WSDL to add port and grid server information" on page 6 describes how to edit the WSDL and change the web service grid server.

Locate the WSDL

The WSDLs for each service can be found in two locations. The WSDLs in the two locations differ only in whether they contain information specific to the grid server (service URL and port number) regarding where the SOAP server can be found by the client. The two WSDL locations are as follows:

- All WSDL documents are linked into the WSDLs grid directory at the top level of the Avaki data catalog. These WSDLs contain the correct grid server and port information:

  ```
  /WSDLs/AvakiServices
  ```

- The WSDLs are also supplied in the following local file:

  ```
  <Avaki-install-dir>/examples/axis/Avaki_ws_client.jar
  ```

  You can use either the Jar or the WinZip utility to unpack the JAR file. To access the WSDLs in the JAR file, you must install Avaki software, but it is not necessary to have a grid server running. However, you must edit WSDLs from the JAR file to add grid server and port information.

Edit the WSDL to add port and grid server information

When you use a WSDL from the JAR file, you must modify the WSDL to replace the placeholder service URL with the DNS name or IP address of the grid server you want the web service to run on. The WSDL can point to any grid server; the server need not be a GDC.

You might also choose to change the grid server or port information in a WSDL from the WSDLs grid directory. (For example, suppose that network restrictions or performance considerations force you to invoke your web service on a foreign grid server, or that you prefer HTTP to HTTPS, or vice versa.)

Be sure to edit a copy of the WSDL, not the original in the JAR or the WSDLs grid directory.

The subsections that follow show how to edit the grid server and port information in a WSDL's service URL.

Editing a WSDL from the JAR file. This example shows a fragment of a WSDL for a document/literal service.

```
<wsdl:service name="AvakiAPIDocLitService">
  <wsdl:port binding="impl:AvakiAPIDocLitSoapBinding"
    name="AvakiAPIDocLit">
   <wsdlsoap:address
     location="https://LOCAL_HOST:8443/axis/services
             /AvakiAPIDocLit"/>
  </wsdl:port>
</wsdl:service>
```

Step 1 (Required.) Change `LOCAL_HOST` to the DNS name or IP address of the grid server machine on which this web service will run. If you're not sure what to enter, look in the grid server's system.properties file for the property java.rmi.server.hostname. Use the value of java.rmi.server.hostname to replace `LOCAL_HOST`. If java.rmi.server.hostname is not set in the system.properties file, simply use the name of the machine on which the grid server is running.

Step 2 If necessary, change the port number (here 8443) to match the actual HTTP or HTTPS port for your grid server. The default port that grid servers use for HTTP is 7080; for HTTPS it is port 8443. To determine what the HTTP and HTTPS ports are for a particular grid server, look in that server's bindings.xml file.

Step 3 Make sure that the protocol indicator in the service URL is of the correct type for the port number. The service URL should begin with "http" if you're using an HTTP port (such as 7080), or with "https" if you're using an HTTPS port (such as 8443).

Editing a WSDL from the data grid. This example shows a fragment of a WSDL for a document/literal service. Notice that it includes the name of the GDC machine, MyGDC.MyCompany.com. It also includes the GDC's actual HTTPS port number (which happens to be the default), 8443.

```
<wsdl:service name="AvakiAPIDocLitService">
  <wsdl:port binding="impl:AvakiAPIDocLitSoapBinding"
    name="AvakiAPIDocLit">
   <wsdlsoap:address
     location="https://MyGDC.MyCompany.com:8443/axis/
             /services/AvakiAPIDocLit"/>
  </wsdl:port>
</wsdl:service>
```

Step 1 If necessary, change the name of the grid server (here, MyGDC.MyCompany.com) to the DNS name or IP address of the grid server machine on which this web service will run. If you're not sure what to enter, look in the grid server's system.properties file for the value of the property java.rmi.server.hostname. If java.rmi.server.hostname is not set in the system.properties file, simply use the name of the machine on which the grid server is running.

Step 2 If necessary, change the port number (here 8443) to match the actual HTTP or HTTPS port for your grid server. The default port that grid servers use for HTTP is 7080; for HTTPS it is port 8443. To determine what the HTTP and HTTPS ports are for a particular grid server, look in that server's bindings.xml file.

Step 3 Make sure that the protocol indicator in the service URL is of the correct type for the port number. The service URL should begin with "http" if you're using an HTTP port (such as 7080), or with "https" if you're using an HTTPS port (such as 8443).

For more information on ports and system properties, see the *Sybase Avaki EII Administration Guide*.

Security with web services

Security is an important consideration in any client/server application. There are several aspects to a complete security solution: privacy and integrity, authentication, and authorization.

Privacy and integrity

The privacy and integrity of the SOAP request/response transmission process is maintained primarily by encryption. Using HTTPS rather than HTTP for SOAP calls provides encryption and protection for your application. In order to use HTTPS, you must configure your SOAP client to trust the SSL certificate that is used by the grid server the client connects to (that is, the grid server on which the web service runs). For example, if your SOAP client is written in Java, you configure it by editing its cacerts file with the **keytool** utility. A sample file, configured to use the default Avaki-signed certificates, can be found in resources/cacerts in the following JAR file:

```
<Avaki-install-dir>/examples/axis/Avaki_ws_client.jar
```

JAR files can be unpacked using either the Jar or the WinZip utility.

SOAP clients of other types handle SSL certificates in different ways. For an explanation of how Microsoft's .NET framework handles SSL certificates, see:

`http://support.microsoft.com/default.aspx?scid=kb;EN-US;Q823177`

In addition, for a code example showing one way to handle certificates in a .NET environment, see:

```
<Avaki-install-dir>/examples/dotnet/AvakiPolicy.vb
```

Authentication and authorization

The WS API provides authentication and authorization in the same manner that the data grid generally handles these aspects of security: through login information and setting access controls. Login information is passed in as a parameter with every client SOAP call in a structure called the `AvakiPrincipal`. The Avaki Principal is described in detail in the section "AvakiPrincipal" on page 14. In order for the web service invocation to be successful, the data grid login account passed into the SOAP request as an Avaki Principal must have the correct access permissions for whatever operation is required. For example, a simple directory listing (`ls`) would only require read permission on that directory, while a database operation request would require at least read and execute permission.

Web services client examples

Avaki provides several sample web service clients written in Java, Perl and VB .NET. The client code examples are located as follows:

- Java examples are in the WS Java archive:

  ```
  <Avaki-install-dir>/examples/Avaki_ws_client.jar
  ```

- Perl examples are in

  ```
  <Avaki-install-dir>/examples/soaplite
  ```

- VB .NET examples are in

  ```
  <Avaki-install-dir>/examples/dotnet
  ```

Chapter 2

Web services API reference

The Avaki web services API (WS API) consists of several WSDL documents that formally specify the legitimate operations, the parameters to the operations and their return values. Avaki provides four WSDL documents as part of the same API. These four WSDL documents are named:

- AvakiAPIDocLit.wsdl
- AvakiAPIRpcEnc.wsdl
- AvakiAPIWithMIMEDocLit.wsdl
- AvakiAPIWithMIMERpcEnc.wsdl

The underlying functionality represented by these WSDLs is nearly identical. The only differences are:

- The DocLit and RpcEnc WSDLs differ solely in the web services style and usage for accessing the services.
- The WithMIME WSDLs contain operations that return bulk results as MIME attachments in addition to operations from the other WSDLs.

Accordingly, this chapter is divided into the following main sections, which are introduced *solely for presentation*. The operations themselves, as described in the WSDLs and as implemented, belong to a single, monolithic API.

- "Complex type descriptions" on page 13: Describes the complex types or data structures used in the WS API.

- “Data catalog operations” on page 18: Describes standard file and directory operations.
- “Data service operations” on page 34: Describes operations that allow you to execute any data service that has been defined in a grid.
- “Database operation operations” on page 41: Describes operations that allow you to execute any database operation (DBOP) that has been defined in a grid.

Each of the operations above is present in each of the WSDLs listed above unless specifically noted otherwise.

Using this reference

The data type and operation descriptions in this reference are independent of programming language and are more easily readable than the formal XML specification in the WSDL documents. Such descriptions necessarily introduce some ambiguity. The WSDL for each service is the definitive authority for Avaki’s SOAP implementation.

Complex type descriptions

Complex types are the custom data structures defined in the WSDL. The WSDL defines array data structures for some of the complex types below. Array data structures are denoted by the suffix `[]` when used.

The entry for each complex type contains the following information:

Name: The name of the complex type as it appears in the WSDL.

Description: A description of the purpose and usage of the type.

Element sequence: When appropriate, a listing of the data elements, their types, and a description of each.

AdHocDBOPExecutionParams

The structure used to pass parameters to an ad-hoc query. Executing an ad-hoc query is similar to defining a database operation and then immediately using it. Thus, all the information to define a database operation must also be provided here.

Element sequence

Element	Type	Description and Values
callable	boolean	Indicates whether the ad-hoc query is a stored procedure.
parameterTypes	string	Contains parameter specifications in standard JDBC format. for example: `in:BIGINT;out:VARCHAR`.
prepared	boolean	Indicates whether the ad-hoc query is a prepared statement.
sql	string	Contains the SQL statement for the ad-hoc query.
values	string[]	Contains the values for the ad-hoc query. The array should have as many elements as required by the SQL string.

AvakiPrincipal

The structure used to pass credential information from the client to the service. The contents of this structure mirror the information needed to log in through the CLI or the web interface. The client must have the permissions required for the requested operation. For example, only read permission is required to list directory contents, but read and execute permissions are required to execute a database operation.

Element sequence

Element	Type	Description and Values
name	string	Login name; case is significant. Alternatively, supply `<name>@<authService>.<authServiceType>.<domain>` in the `name` field and provide nil values for all fields except `password`.
password	string	Password; case is significant.
authService	string	Pathname of the authentication service; when a nil value is passed, `DefaultAuthService` will be used. Case is significant.
authServiceType	string	Values can be: `Grid`, `Nis`, or `Ldap`; when a nil value is passed, `Grid` will be used. Case is significant.
domain	string	Avaki domain name; when a nil value is passed, the server's domain name will be used. Case is significant.

DataCatalogAttribute

The structure that represents a data catalog attribute. For more information on custom attributes and legal types see the *Sybase Avaki EII Administration Guide*.

Element sequence

Element	Type	Description and Values
name	string	Name of the attribute.
value	string	Value of the attribute.
type	string	Type of the attribute. Legal types are: `String`, `Integer`, `Float`, `Date`, `Time`, and `Timestamp`.

DataCatalogEntry

The structure that represents an entry in the data catalog.

Element sequence

Element	Type	Description and Values
basename	string	The target grid catalog entry name.
fullpath	string	The full pathname including the grid catalog entry name referenced by the base-name.
type	string	Avaki directory entry types including: directory, file, share, directory and object.

DataCatalogPermission

The structure that represents a permission on an entry in the data catalog.

Element sequence

Element	Type	Description and Values
subject	string	The user or group name for which the permission applies.
subjectGroup	boolean	An indicator of whether the subject is a group or not.
action	string	The action for which the permission applies. Legal values are `read`, `write`, `execute` and `delete`.
constraint	string	The constraint on the action. Legal values are `allow` and `deny`.
owner	boolean	An indicator of whether the subject is the owner of the entry for which this permission applies.
groupOwner	boolean	An indicator of whether a group to which the subject belongs is the owner of the entry for which this permission applies.

DataServiceExecutionParams

The structure used to pass data service execution parameters. The two string arrays must be of equal length and have exactly the number of elements as the parameters required by the data service.

Element sequence

Element	Type	Description and Values
names	string[]	Names of the parameters expected by the data service.
values	string[]	Values of the parameters expected by the data service.

DBOPExecutionParams

The structure used to pass database operation execution parameters. The number of elements in the array must match the number of parameters in the database operation.

Element sequence

Element	Type	Description and Values
values	string[]	The parameter values expected by the database operation.

SearchQuery

The structure that represents a query to a search service. For information on custom attributes and legal types, see the *Sybase Avaki EII Administration Guide*.

Element sequence

Element	Type	Description and Values
name	string	Name of the search term.
value	string	The optional value of the term to search.
expression	string	The optional expression to use for searching. Legal values are `=`, `>`, `<`, `>=`, `<=` and `<>`. The default is `=`.
valueType	string	The optional type of the term. Legal types are: `String`, `Integer`, `Float`, `Date`, `Time`, and `Timestamp`.
snippetType	string	The optional kind of the term. Currently, this value is unused because the search always searches on attributes.

SearchResult

The structure that represents the result of a query to a search service. For information on custom attributes and legal types see the *Sybase Avaki EII Administration Guide*.

Element sequence

Element	Type	Description and Values
fullpath	string	The data catalog entry that satisfies the search query.
pathType	string	Avaki directory entry types including directory, file, share, directory and object.
snippet	string	The term and value that satisfies the search query.
snippetType	string	The optional kind of the term. Currently, this value is always `attribute`.
snippetSubType	string	The optional type of the snippet. Legal types are `String`, `Integer`, `Float`, `Date`, `Time`, and `Timestamp`.

Data catalog operations

This section contains standard file and directory operations. Each operation is described below with its expected call signature and return value.

For more information on the data catalog, see the *Sybase Avaki EII Overture*.

Client code sample

This small example is written in Java. It assumes you are using Axis as your web services infrastructure on the client side. Specifically, doing so assumes you have taken the Avaki WSDL, run Axis's wsdl2java tool to generate stubs, and written your Java client using those stubs. In this example, we assume those stubs are in the package `com.avaki.ws.stubs`.

Using the data catalog API

```
import com.avaki.ws.stubs.AvakiAPIRpcEncServiceLocator;
import com.avaki.ws.stubs.AvakiAPIRpcEnc;
import com.avaki.api.common.*;

public class WSSampleDCClient
```

```
{
    AvakiPrincipal principal = new AvakiPrincipal();
    principal.setName(...);
    principal.setPassword(...);
    principal.setAuthService(...);
    principal.setAuthServiceType(...);
    principal.setDomain(...);
    // Look up the service using Axis-generated stubs.
    AvakiAPIRpcEncServiceLocator locator =
        new AvakiAPIRpcEncServiceLocator();
    AvakiAPIRpcEnc remote = locator.getAvakiAPIRpcEnc();
    String[] results = remote.whoami(principal);
    ...
}
```

accessiblePath

Determines whether the specified catalog entry is accessible to the requesting user specified by the `AvakiPrincipal`.

Request signature

`accessiblePath(<principal>, <path>)`

Arguments	Type	Description
principal	AvakiPrincipal	Required authentication information.
path	string	The full pathname of the catalog entry.

Returns

Returns a boolean indicating whether the catalog entry specified by `path` could be accessed. Either a nonexistent catalog entry or insufficient access permissions for the supplied `principal` will return `false`.

chmod

Sets a permission on the data catalog entry. For information about setting the owner of a data catalog query, see the "chown" section, below.

Request signature

```
chmod(<principal>, <path>, <permission>, <recursive>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
permission	DataCatalog-Permission	The subject, action and constraint of the permission to set.
recursive	boolean	Specifies whether to set permissions on directories recursively.

Returns

Returns a boolean indicating whether the permission was set successfully.

chown

Sets the owner of a data catalog entry. For information about setting a permission on a data catalog entry, see "chmod," above.

Request signature

```
chown(<principal>, <path>, <user>, <recursive>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
user	string	The new owner of the catalog entry.
recursive	boolean	Specifies whether to change ownership on directories recursively.

Returns

Returns a boolean indicating whether the ownership was changed successfully or not.

fileRead

Reads the specified portion of the file.

Request signature

```
fileRead(<principal>, <path>, <offset>, <count>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
offset	long	Index in bytes of the first entry to return. The index is 0-based.
count	integer	Number of bytes to return.

Returns

Returns a byte-array containing `count` number of bytes, starting at `offset` number of bytes into the file.

Note When this operation is executed from the rpc/encoded WSDLs (AvakiAPI-RpcEnc.wsdl and AvakiAPIWithMIMERpcEnc.wsdl), the results are returned as a base-64 encoded sequence of bytes. Your client infrastructure must decode them in order to view the results. When this operation is executed from the document/literal WSDLs (AvakiAPIDocLit.wsdl and AvakiAPIWithMIME-DocLit.wsdl), each byte of the results is returned in its own tag, thus increasing the amount of data transferred greatly.

fileReadAttach

Reads the specified portion of the file and returns the result as a SOAP attachment.

Request signature

`fileReadAttach(<principal>, <path>, <offset>, <count>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
offset	long	Index in bytes of the first entry to return. The index is 0-based.
count	integer	Number of bytes to return.

Returns

Returns a SOAP attachment containing `count` number of bytes, starting at `offset` number of bytes into the file.

Note This operation is present only in the WSDLs that support MIME attachments, AvakiAPIWithMIMEDocLit.wsdl and AvakiAPIWithMIMERpcEnc.wsdl.

fileReadString

Reads the specified portion of the file and returns the results as a string.

Request signature

`fileReadString(<principal>, <path>, <offset>, <count>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
offset	long	Index in bytes of the first entry to return. The index is 0-based.
count	integer	Number of bytes to return.

Returns

Returns a Unicode character string containing `count` number of characters, starting at `offset` number of bytes into the file.

fileWrite

Writes to the file from the byte buffer. If the file does not exist it will be created at the directory location specified in `path`.

Request signature

```
fileWrite(<principal>, <path>, <offset>, <buf>, <count>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
offset	long	Index in bytes of the first location to write into in the target file. The index is 0-based.
buf	byte-array	Buffer containing data to be written to the file.
count	integer	Number of bytes to write.

Returns

Returns an integer indicating the number of bytes written to the file. Can be used to test whether the `fileWrite` operation was successful and complete.

getAttributes

Gets the system and user-defined attributes of the specified data catalog entry. System attributes are read-only attributes that are provided by default for every object in the grid. User attributes are customizable attributes that can be created for a grid file, directory, share, server, or service.

Request signature

```
getAttributes(<principal>, <path>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.

Returns

Returns a `DataCatalogAttribute[]`.

getSystemAttributes

Gets only the system-level attributes of the specified data catalog entry.

Request signature

```
getSystemAttributes(<principal>, <path>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.

Returns

Returns a `DataCatalogAttribute[]`.

getUserAttributes

Gets only the user-level attributes of the specified data catalog entry.

Request signature

```
getUserAttributes(<principal>, <path>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.

Returns

Returns a `DataCatalogAttribute[]`.

listDomains

Lists all the domains linked to the current domain.

Request signature

```
listDomains(<principal>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.

Returns

Returns an array of strings containing the name of each domain.

listSearches

Lists all the search services in the specified domain.

Request signature

```
listSearches(<principal>, <domain>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
domain	string	Name of a domain. If this value is `null`, the current domain is assumed.

Returns

Returns an array of strings containing the name of each search service.

ls

Lists the contents of the grid directory specified in the `path` argument, inclusive of the specified indices.

Request signature

```
ls(<principal>, <path>, <from>, <to>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
from	integer	Index of the first entry to return. Index is 1-based.
to	integer	Index of the last entry to return.

Returns

Returns a `DataCatalogEntry[]` containing the elements requested.

lsSize

Returns the number of entries in the grid directory specified in the `path` argument, inclusive of the `"."` and `".."` entries.

Request signature

```
lsSize(<principal>, <path>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.

Returns

Returns an integer indicating the number of entries in the requested grid directory.

mkdir

Creates a grid directory.

Request signature

```
mkdir(<principal>, <path>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the directory to create.

Returns

Returns a boolean indicating whether directory creation was successful. If the grid directory already exists, `false` is returned.

mkdirParents

Creates a grid directory and any parent directories required.

Request signature

```
mkdirParents(<principal>, <path>, <makeParents>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the directory to create.
makeParents	boolean	Specifies whether to create the parent directory if not present.

Returns

Returns a boolean indicating whether directory creation was successful. If the grid directory already exists, creation is considered successful and no error is returned.

mkdirParentsServer

Creates a grid directory and any parent directories required on the specified server.

Request signature

```
mkdirParentsServer(<principal>, <path>, <makeParents>,
  <serverPath>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the directory to create.
makeParents	boolean	Specifies whether to create the parent directory if not present.
serverPath	string	The full pathname of the server where the directory should be created. The pathname format is /System/LocalDomain/Servers/<grid-server-name>/Server.

Returns

Returns a boolean indicating whether directory creation was successful. If the grid directory already exists, creation is considered successful and no error is returned.

mkdirServer

Creates a grid directory on the specified server.

Request signature

```
mkdirServer(<principal>, <path>, <serverPath>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the directory to create.
serverPath	string	The full pathname of the server where the directory should be created. The pathname format is /System/LocalDomain/Servers/<grid-server-name>/Server.

Returns

Returns a boolean indicating whether directory creation was successful. If the grid directory already exists, `false` is returned.

mv

Moves an object in the data catalog (a grid directory or a shared file, for example) to the specified target directory.

Request signature

```
mv(<principal>, <srcPath>, <destPath>, <force>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
srcPath	string	The full pathname of the catalog object to move.
destPath	string	The full pathname of the destination grid directory. When a file is being moved and the filename is included in the destination path, the moved file will be renamed.
force	boolean	Indicates whether to force a move of protected system files or directories. The operation's `principal` must have correct access permissions for the operation.

Returns

Returns a boolean indicating whether the move was successful.

permissions

Gets the permissions on the specified data catalog entry. For information about setting a permission on a data catalog entry, see "chmod" on page 20. For information about setting the owner of a data catalog query, see "chown" on page 20.

Request signature

```
permissions(<principal>, <path>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.

Returns

Returns a `DataCatalogPermission[]`.

removeAttribute

Removes a user attribute from the specified data catalog entry. For information about getting the attributes of a data catalog entry, see "getAttributes" on page 24. For information about setting an attribute, see "setAttribute" on page 33.

Request signature

```
removeAttribute(<principal>, <path>, <attribute>, <recursive>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
attribute	DataCatalog-Attribute	The name and type of the attribute to remove.
recursive	boolean	Specifies whether to set attributes on directories recursively.

Returns

Returns a boolean indicating whether the attribute was removed successfully or not.

rm

Removes the data catalog entry specified in the `path` argument.

Request signature

```
rm(<principal>, <path>, <recursive>, <force>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
recursive	boolean	Specifies whether to remove directories recursively.

Arguments	Type	Description
force	boolean	Indicates whether to force a deletion of protected system files or directories. The operation's `principal` must have correct access permissions for the operation. When `force` is true, catalog entries are ignored if nonexistent, and removed if present but corrupted.

Returns

Returns a boolean indicating whether the data catalog entry was removed successfully. When entries are missing or corrupted, the `force` parameter determines whether the operation is considered successful.

search

Invokes a search operation to return the results matching a comparison.

Request signature

`search(<principal>, <service>, <query>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
service	string	The pathname or qualified name of the search service. The format for the qualified name is: `<domain-name>.<search-service-name>`.
query	SearchQuery	The name, value, type and expression for the search query.

Returns

Returns a `SearchResult[]` containing a list of matches for the query along with the content that satisfies the expression in the query.

setAttribute

Sets a user attribute on the specified data catalog entry. For information about getting the attributes of a data catalog entry, see "getAttributes" on page 24. For information about removing an attribute, see "removeAttribute" on page 31.

Request signature

```
setAttribute(<principal>, <path>, <attribute>, <recursive>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
path	string	The full pathname of the catalog entry.
attribute	`DataCatalogAttribute`	The name, type and value of the attribute to set.
recursive	boolean	Specifies whether to set attributes on directories recursively.

Returns

Returns a boolean indicating whether the attribute was set successfully.

tester

An operation that you can use to test whether your web services client is working properly independent of any Avaki operations.

Request signature

```
tester(<in>)
```

Arguments	Type	Description
in	string	Any string you want to use for testing.

Returns

Echoes the string supplied by the request.

whoami

Provides standard information about the login represented by the supplied `Avaki-Principal`.

Request signature

`whoami(<principal>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.

Returns

Returns an array of strings containing information about the web services client login, including username, groupname, and associated authentication service.

Data service operations

The data services section contains operations that allow you to execute any data service that has been defined in a grid. Each operation is described below with its expected call signature and return value.

Executing a data service typically consists of the following steps:

Step 1 Run the method executeDS (page 37) and get a token back.

Step 2 (Optional) Run isDSAvakiXML (page 40) on the data service if you need to determine whether the result set associated with the token is in the Avaki XML rowset format.

Step 3 Run one of the following methods:

- getDSOutput (page 37)
- getDSOutputAttach (page 38)
- getDSOutputString (page 39)

These methods are described in the sections that follow.

Several of the SOAP operations in this section return data using, or require data in, the Avaki XML rowset format. For more information on the Avaki XML rowset format and data services see the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*.

Several of the SOAP operations in this section require the `dsName` argument. The `dsName` argument can be specified as either a full pathname or as a qualified name of this form: `<domain-name>.<data-service-name>`.

Client code sample

This small example is written in Java. It assumes you are using Axis as your web services infrastructure on the client side. Specifically, doing so assumes you have taken the Avaki WSDL, run Axis's wsdl2java tool to generate stubs, and written your Java client using those stubs. In this example, we assume those stubs are in the package `com.avaki.ws.stubs`.

Using the data service API

```
import com.avaki.ws.stubs.AvakiAPIRpcEncServiceLocator;
import com.avaki.ws.stubs.AvakiAPIRpcEnc;
import com.avaki.api.common.*;

public class WSSampleDSClient
{
     AvakiPrincipal principal = new AvakiPrincipal();
     principal.setName(...);
     principal.setPassword(...);
     principal.setAuthService(...);
     principal.setAuthServiceType(...);
     principal.setDomain(...);
     // Look up the service using Axis-generated stubs.
     AvakiAPIRpcEncServiceLocator locator =
         new AvakiAPIRpcEncServiceLocator();
     AvakiAPIRpcEnc remote = locator.getAvakiAPIRpcEnc();
     String dsName = ...;
     String[] paramNames = new String[...];
     String[] paramStrs = new String[...];
     for (int i = 0; i < paramStrs.length; i++)
     {
         paramNames[i] = ...;
         paramStrs[i] = ...;
     }
     DataServiceExecutionParams params = new DataServiceExecutionParams();
```

```
    params.setNames(paramNames);
    params.setValues(paramStrs);
    int token = remote.executeDS(principal, dsName, params);
    int maxBytes = 8 * 1024;
    int isAvakiXML = remote.isDSAvakiXML(principal, token);
    byte[] results = remote.getDSOutput(principal, token, maxBytes,
        isAvakiXML);
    ...
}
```

accessibleDS

Determines whether the specified data service is accessible to the requesting user specified by the `AvakiPrincipal`.

Request signature

`accessibleDS(<principal>, <dsName>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dsName	string	The full pathname or qualified name of the data service.

Returns

Returns a boolean indicating whether the data service specified by `dsName` could be accessed. Either a nonexistent data service or insufficient access permissions for the supplied `principal` will return `false`.

executeDS

Executes the specified data service.

Request signature

```
executeDS(<principal>, <dsName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dsName	string	The full pathname or qualified name of the data service.
parameters	DataServiceEx-ecutionParams	The execution parameters required by this data service.

Returns

Returns an integer, which is a token or handle to use for obtaining the results of the operation.

getDSOutput

Gets the output from the `executeDS` operation.

Request signature

```
getDSOutput(<principal>, <token>, <maxRowsBytes>,
  <isDSAvakiXML>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `executeDS` operation used as a handle to access the results.
maxRowsBytes	integer	Integer specifying the maximum number of rows or bytes to return. When `isDSAvakiXML` is `true`, `maxRowsBytes` refers to rows; when `isDSAvakiXML` is `false`, `maxRowsBytes` refers to bytes.
isDSAvakiXML	boolean	Boolean indicating whether the result set associated with the token is in the Avaki XML rowset format.

Returns

Returns a byte-array containing the requested results of the data service operation. When `isDSAvakiXML` is `true`, returned results will be in valid Avaki XML rowset format containing at most the requested number of rows. When `isDSAvakiXML` is `false` the results will be at most the number of bytes requested.

Caution Invoke this operation *only* if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

getDSOutputAttach

Gets the output from the `executeDS` operation and returns it as a SOAP attachment.

Request signature

```
getDSOutputAttach(<principal>, <token>, <maxRowsBytes>,
  <isDSAvakiXML>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `executeDS` operation used as a handle to access the results.
maxRowsBytes	integer	Integer specifying the maximum number of rows or bytes to return. When `isDSAvakiXML` is `true`, `maxRowsBytes` refers to rows; when `isDSAvakiXML` is `false`, `maxRowsBytes` refers to bytes.
isDSAvakiXML	boolean	Boolean indicating whether the result set associated with the token is in the Avaki XML rowset format.

Returns

Returns a SOAP attachment containing the requested results of the data service operation. When `isDSAvakiXML` is `true`, returned results will be in valid Avaki XML rowset format containing at most the requested number of rows. When `isDSAvakiXML` is `false` the results will be at most the number of bytes requested.

Note This operation is present only in the WSDLs that support MIME attachments, AvakiAPIWithMIMEDocLit.wsdl and AvakiAPIWithMIMERpcEnc.wsdl.

getDSOutputString

Gets the output from the `executeDS` operation.

Request signature

```
getDSOutputString(<principal>, <token>, <maxRowsBytes>,
  <isDSAvakiXML>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `executeDS` operation used as a handle to access the results.
maxRowsBytes	integer	Integer specifying the maximum number of rows or bytes to return. When `isDSAvakiXML` is `true`, `maxRowsBytes` refers to rows; when `isDSAvakiXML` is `false`, `maxRowsBytes` refers to bytes.
isDSAvakiXML	boolean	Boolean indicating whether the result set associated with the token is in the Avaki XML rowset format.

Returns

Returns a Unicode character string containing the requested results of the data service operation. When `isDSAvakiXML` is `true`, returned results will be in valid Avaki XML rowset format containing at most the requested number of rows. When `isDSAvakiXML` is `false` the results will be at most the number of bytes requested.

getDSParameters

Gets the parameters of the specified data service.

Request signature

```
getDSParameters(<principal>, <dsName>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dsName	string	The full pathname or qualified name of the data service.

Returns

Returns an array of strings containing the parameter names of the data service.

isDSAvakiXML

Determines whether a data service result is in the Avaki XML rowset format.

Request signature

```
isDSAvakiXML(<principal>, <token>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `executeDS` operation used as a handle to access the results.

Returns

Returns a boolean indicating whether the result set associated with `token` is in the Avaki XML rowset format.

listDSs

Lists all the data services in the specified domain.

Request signature

```
listDSs(<principal>, <domain>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
domain	string	Name of a domain. If this value is `null`, the current domain is assumed.

Returns

Returns an array of strings containing the full pathname of each data service.

Database operation operations

The database operations section contains operations that allow you to execute any database operation or ad-hoc query that has been defined in a grid. An ad-hoc query works like a database operation that is executed immediately after you define it. Each SOAP operation is described below with its expected call signature and return value.

Several of the SOAP operations in this section return data using, or require data in, the Avaki XML rowset format. For more information about the Avaki XML rowset format and database operations, see the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*.

Several of the SOAP operations in this section require the `dbopName` argument. The `dbopName` argument can be specified as either a full pathname or as a JDBC-format qualified name, `<domain-name>.<dbconn-name>.<dbop-name>`.

The typical procedure for executing a database operation or ad-hoc query is as follows:

Step 1 Run one of the following methods and get a token back:

- For database operations:
  - executeDBOp (page 49)
  - executeDBOpWithOutput (page 50)
  - executeDBOpWithOutputAttach (page 52) or executeDBOpWithOutputString (page 53)
- For ad-hoc queries:
  - executeAdHocDBOp (page 44)
  - executeDBOpBytesInput (page 49)
  - executeDBOpGridFileInput (page 50)
  - executeAdHocDBOpWithOutput (page 45)
  - executeAdHocDBOpWithOutputAttach (page 46) or executeAdHocDBOpWithOutputString (page 47)

Step 2 Run one of the following methods:

- getDBOpOutput (page 54)

- getDBOpOutputAttach (page 55)
- getDBOpOutputString (page 56)

These methods are described in the sections that follow.

Client code sample

This small example is written in Java. It assumes you are using Axis as your web services infrastructure on the client side. Specifically, doing so assumes you have taken the Avaki WSDL, run Axis's wsdl2java tool to generate stubs, and written your Java client using those stubs. In this example, we assume those stubs are in the package `com.avaki.ws.stubs`.

Using the database operations API

```
import com.avaki.ws.stubs.AvakiAPIRpcEncServiceLocator;
import com.avaki.ws.stubs.AvakiAPIRpcEnc;
import com.avaki.api.common.*;

public class WSSampleDBOpClient
{
     AvakiPrincipal principal = new AvakiPrincipal();
     principal.setName(...);
     principal.setPassword(...);
     principal.setAuthService(...);
     principal.setAuthServiceType(...);
     principal.setDomain(...);
     // Look up the service using Axis-generated stubs.
     AvakiAPIRpcEncServiceLocator locator =
         new AvakiAPIRpcEncServiceLocator();
     AvakiAPIRpcEnc remote =
         locator.getAvakiAPIRpcEnc();
     String dbopName = ...;
     String[] paramStrs = new String[...];
     for (int i = 0; i < paramStrs.length; i++)
     {
         paramStrs[i] = ...;
     }
     DBOPExecutionParams params = new DBOPExecutionParams();
     params.setValues(paramStrs);
     int token = remote.executeDBOp(principal, dbopName, params);
     int maxRows = 100;
     byte[] results = remote.getDBOpOutput(principal, token, maxRows);
     ...
}
```

accessibleDBOp

Determines whether the specified database operation is accessible to the requesting user specified by the `AvakiPrincipal`.

Note The `accessibleDBOp` operation in the database operations API does not work with ad-hoc queries. Use the `accessiblePath` operation in the data catalog section and specify the full pathname to the database connector.

Request signature

```
accessibleDBOp(<principal>, <dbopName>)
```

Arguments	**Type**	**Description**
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.

Returns

Returns a boolean indicating whether the database operation specified by `dbopName` could be accessed. Either a nonexistent database operation or insufficient access permissions for the supplied `principal` will return `false`.

executeAdHocDBOp

Executes the specified ad-hoc query. An ad-hoc query works like a database operation that is used immediately after you define it. Therefore, all the information to define a database operation must also be provided for an ad-hoc query.

Request signature

```
executeAdHocDBOp(<principal>, <dbciName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbciName	string	The pathname or qualified name of the database connector. The format for the qualified name is `<domain-name>.<dbconnector-name>`
parameters	AdHocDBOP-Execution-Params	The execution parameters required for this ad-hoc query.

Returns

Returns an integer, which is a token or handle to use for obtaining the results of the operation.

executeAdHocDBOpWithOutput

Executes the specified ad-hoc query and returns all of the output. An ad-hoc query is a database operation that is executed immediately after you define it. Therefore, all the information needed to define a database operation must also be provided here.

Request signature

```
executeAdHocDBOpWithOutput(<principal>, <dbciName>,
  <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbciName	string	The pathname or qualified name of the database connector. The format for the qualified name is `<domain-name>.<dbconnector-name>`
parameters	AdHocDBOP-Execution-Params	The execution parameters required for this ad-hoc query.

Returns

Returns a byte-array containing the requested results of the ad-hoc query.

Caution Invoke this operation *only* if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

If executing an ad-hoc query results in large amounts of output, the following situations can occur:

- Your client will have to wait until the last byte of data is extracted from the back end before it can receive the first byte, resulting in high latency.
- Your connection from the client to the server may time out if latency is too high.
- Large result sets could exhaust memory on the grid server, resulting in a crash. If a crash occurs, you might have to restart the grid server.
- Converting large result sets into XML might cause the convertor to run out of memory, resulting in a crash.

- The SOAP infrastructure on the server side might run out of memory when it serializes the results, resulting in a crash.
- The SOAP infrastructure on the client side might run out of memory when it parses the response and de-serializes the results, resulting in an error.
- The client application might run out of memory when it receives large amounts of XML data.

executeAdHocDBOpWithOutputAttach

Executes the specified ad-hoc query and returns all of the output as a SOAP attachment. An ad-hoc query works like a database operation that is executed immediately after you define it. Therefore, all the information needed to define a database operation must also be provided here.

Request signature

```
executeAdHocDBOpWithOutputAttach(<principal>, <dbciName>,
  <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbciName	string	The pathname or qualified name of the database connector. The format for the qualified name is `<domain-name>.<dbconnector-name>`
parameters	AdHocDBOP-Execution-Params	The execution parameters required for this ad-hoc query.

Returns

Returns a SOAP attachment containing the requested results of the ad-hoc query.

Note This operation is present only in the WSDLs that support MIME attachments, AvakiAPIWithMIMEDocLit.wsdl and AvakiAPIWithMIMERpcEnc.wsdl.

Caution Unrestricted execution of this operation is dangerous.

Invoke this operation only if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

If executing an ad-hoc query results in large amounts of output, the following situations can occur:

- Your client will have to wait until the last byte of data is extracted from the back end before it can receive the first byte, resulting in high latency.
- Your connection from the client to the server may time out if latency is too high.
- Large result sets could exhaust memory on the grid server, resulting in a crash. If a crash occurs, you might have to restart the grid server.
- Converting large result sets into XML might cause the convertor to run out of memory, resulting in a crash.
- The SOAP infrastructure on the server side might run out of memory when it serializes the results, resulting in a crash.
- The SOAP infrastructure on the client side might run out of memory when it parses the response and de-serializes the results, resulting in an error.
- The client application might run out of memory when it receives large amounts of XML data.

executeAdHocDBOpWithOutputString

Executes the specified ad-hoc query and returns all of the output as a character string. An ad-hoc query works like a database operation that is used immediately after you define it. Therefore, all the information to define a database operation must also be provided.

Request signature

```
executeAdHocDBOpWithOutputString(<principal>, <dbciName>,
  <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.

Arguments	Type	Description
dbciName	string	The pathname or qualified name of the database connector. The format for the qualified name is `<domain-name>.<dbconnector-name>`
parameters	AdHocDBOP-Execution-Params	The execution parameters required for this ad-hoc query.

Returns

Returns a Unicode character string containing the requested results of the ad-hoc query.

Caution Unrestricted execution of this operation is dangerous.

Invoke this operation only if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

If executing an ad-hoc query results in large amounts of output, the following situations can occur:

- Your client will have to wait until the last byte of data is extracted from the back end before it can receive the first byte, resulting in high latency.
- Your connection from the client to the server may time out if latency is too high.
- Large result sets could exhaust memory on the grid server, resulting in a crash. If a crash occurs, you might have to restart the grid server.
- Converting large result sets into XML might cause the convertor to run out of memory, resulting in a crash.
- The SOAP infrastructure on the server side might run out of memory when it serializes the results, resulting in a crash.
- The SOAP infrastructure on the client side might run out of memory when it parses the response and de-serializes the results, resulting in an error.
- The client application might run out of memory when it receives large amounts of XML data.

executeDBOp

Executes the specified database operation.

Request signature

```
executeDBOp(<principal>, <dbopName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecu-tionParams	The execution parameters required for this database operation

Returns

Returns an integer, which is a token or handle to use for obtaining the results of the operation.

executeDBOpBytesInput

Executes the specified database operation. The parameters for the database operation are in the `xmlDoc` byte-array whose contents must be in Avaki XML rowset format.

Request signature

```
executeDBOpBytesInput(<principal>, <dbopName>, <xmlDoc>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
xmlDoc	byte-array	Byte-array containing parameters for this database operation; this must be in Avaki XML rowset format.

Returns

Returns an integer, which is a token or handle to use for obtaining the results of the operation.

executeDBOpGridFileInput

Executes the specified database operation. The parameters for the database operation, if any, must be in a data catalog file whose contents are in Avaki XML rowset format. (For details on Avaki XML rowset format, see the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*.)

Request signature

```
executeDBOpGridFileInput(<principal>, <dbopName>,
  <gridFileName>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
gridFileName	string	Pathname of the file in the data catalog containing parameters for this database operation. File must be in the Avaki XML rowset format.

Returns

Returns an integer, which is a token or handle to use for obtaining the results of the operation.

executeDBOpWithOutput

Executes the specified database operation and returns all of the output.

Request signature

```
executeDBOpWithOutput(<principal>, <dbopName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.

Arguments	Type	Description
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecutionParams	The execution parameters required for this database operation

Returns

Returns a byte-array containing the requested results of the database operation.

Note When this operation is executed from the rpc/encoded WSDLs (AvakiAPIRpcEnc.wsdl and AvakiAPIWithMIMERpcEnc.wsdl), the results are returned as a base-64 encoded sequence of bytes. Your client infrastructure must decode them in order to view the results. When this operation is executed from the document/literal WSDLs (AvakiAPIDocLit.wsdl and AvakiAPIWithMIMEDocLit.wsdl), each byte of the results is returned in its own tag, thus increasing the amount of data transferred greatly.

Caution Unrestricted execution of this operation is dangerous.

Invoke this operation only if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

If executing a database operation results in large amounts of output, the following situations can occur:

- Your client will have to wait until the last byte of data is extracted from the back end before it can receive the first byte, resulting in high latency.
- Your connection from the client to the server may time out if latency is too high.
- Large result sets could exhaust memory on the grid server, resulting in a crash. If a crash occurs, you might have to restart the grid server.
- Converting large result sets into XML might cause the convertor to run out of memory, resulting in a crash.
- The SOAP infrastructure on the server side might run out of memory when it serializes the results, resulting in a crash.
- The SOAP infrastructure on the client side might run out of memory when it parses the response and de-serializes the results, resulting in an error.

- The client application might run out of memory when it receives large amounts of XML data.

executeDBOpWithOutputAttach

Executes the specified database operation and returns all of the output as a SOAP attachment.

Request signature

```
executeDBOpWithOutputAttach(<principal>, <dbopName>,
  <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecu-tionParams	The execution parameters required for this database operation

Returns

Returns a SOAP attachment containing the requested results of the database operation.

Note This operation is present only in the WSDLs that support MIME attachments, AvakiAPIWithMIMEDocLit.wsdl and AvakiAPIWithMIMERpcEnc.wsdl.

Caution Unrestricted execution of this operation is dangerous.

Invoke this operation only if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

If executing a database operation results in large amounts of output, the following situations can occur:

- Your client will have to wait until the last byte of data is extracted from the back end before it can receive the first byte, resulting in high latency.
- Your connection from the client to the server may time out if latency is too high.

- Large result sets could exhaust memory on the grid server, resulting in a crash. If a crash occurs, you might have to restart the grid server.
- Converting large result sets into XML might cause the convertor to run out of memory, resulting in a crash.
- The SOAP infrastructure on the server side might run out of memory when it serializes the results, resulting in a crash.
- The SOAP infrastructure on the client side might run out of memory when it parses the response and de-serializes the results, resulting in an error.
- The client application might run out of memory when it receives large amounts of XML data.

Invoke this operation only if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

executeDBOpWithOutputString

Executes the specified database operation and returns all of the output as a character string.

Request signature

```
executeDBOpWithOutputString(<principal>, <dbopName>,
  <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecutionParams	The execution parameters required for this database operation

Returns

Returns a Unicode character string containing the requested results of the database operation.

Caution Unrestricted execution of this operation is dangerous.

Invoke this operation only if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

If executing a database operation results in large amounts of output, the following situations can occur:

- Your client will have to wait until the last byte of data is extracted from the back end before it can receive the first byte, resulting in high latency.
- Your connection from the client to the server may time out if latency is too high.
- Large result sets could exhaust memory on the grid server, resulting in a crash. If a crash occurs, you might have to restart the grid server.
- Converting large result sets into XML might cause the convertor to run out of memory, resulting in a crash.
- The SOAP infrastructure on the server side might run out of memory when it serializes the results, resulting in a crash.
- The SOAP infrastructure on the client side might run out of memory when it parses the response and de-serializes the results, resulting in an error.
- The client application might run out of memory when it receives large amounts of XML data.

getDBOpOutput

Gets the output from a database operation.

Request signature

```
getDBOpOutput(<principal>, <token>, <maxRows>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `execute-` operation used as a handle to access the results.
maxRows	integer	Integer specifying the maximum number of rows to return.

Returns

Returns a byte-array containing the requested results of the database operation.

Note When this operation is executed from the rpc/encoded WSDLs (Avaki-APIRpcEnc.wsdl and AvakiAPIWithMIMERpcEnc.wsdl), the results are returned as a base-64 encoded sequence of bytes. Your client infrastructure must decode them in order to view the results. When this operation is executed from the document/literal WSDLs (AvakiAPIDocLit.wsdl and AvakiAPIWithMIME-DocLit.wsdl), each byte of the results is returned in its own tag, thus increasing the amount of data transferred greatly.

getDBOpOutputAttach

Gets the output from a database operation and returns it as a SOAP attachment.

Request signature

```
getDBOpOutputAttach(<principal>, <token>, <maxRows>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `execute-` operation used as a handle to access the results.
maxRows	integer	Integer specifying the maximum number of rows to return.

Returns

Returns a SOAP attachment containing the requested results of the database operation.

Note This operation is present only in the WSDLs that support MIME attachments, AvakiAPIWithMIMEDocLit.wsdl and AvakiAPIWithMIMERpcEnc.wsdl.

getDBOpOutputString

Gets the output from a database operation and returns it as a character string.

Request signature

`getDBOpOutputString(<principal>, <token>, <maxRows>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
token	integer	The integer returned by the `execute-` operation used as a handle to access the results.
maxRows	integer	Integer specifying the maximum number of rows to return.

Returns

Returns a Unicode character string containing the requested results of the database operation.

getDBOpParameters

Gets the parameters used by the specified database operation.

Request signature

`getDBOpParameters(<principal>, <dbopName>)`

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.

Returns

Returns an array of strings containing the parameters of this database operation. For example, the first location in the array might contain the string in:VARCHAR and the next the string out:BIGINT, and so on.

getDBOpSchema

Executes the specified database operation and returns only its schema.

Request signature

```
getDBOpSchema(<principal>, <dbopName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecu-tionParams	The execution parameters required for this database operation

Returns

Returns a byte-array containing the XML schema for the database operation.

Caution Invoke this operation *only* if you are sure that the results of the operation will not exhaust memory and network resources on the client or the server.

getDBOpSchemaAttach

Executes the specified database operation and returns only its schema as a SOAP attachment.

Request signature

```
getDBOpSchemaAttach(<principal>, <dbopName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecu-tionParams	The execution parameters required for this database operation

Returns

Returns a SOAP attachment containing the XML schema for the database operation.

Note This operation is present only in the WSDLs that support MIME attachments, AvakiAPIWithMIMEDocLit.wsdl and AvakiAPIWithMIMERpcEnc.wsdl.

getDBOpSchemaString

Executes the specified database operation and returns only its schema as a character string.

Request signature

```
getDBOpSchemaString(<principal>, <dbopName>, <parameters>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbopName	string	The full pathname or qualified name of the database operation.
parameters	DBOPExecutionParams	The execution parameters required for this database operation

Returns

Returns a Unicode character string containing the XML schema for the database operation.

getSQL

Gets the SQL statement defined for this database operation.

Request signature

```
getSQL(<principal>, <dbopName>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.

Arguments	Type	Description
dbopName	string	The full pathname or qualified name of the database operation.

Returns

Returns a Unicode character string containing the SQL statement for this database operation.

listDBConns

Lists all the database connectors in the specified domain.

Request signature

```
listDBConns(<principal>, <domain>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
domain	string	Name of a domain. If this value is `null`, the current domain is assumed.

Returns

Returns an array of strings containing the full pathname of each database connector.

listDBOps

Lists all the database operations in the specified domain.

Request signature

```
listDBOps(<principal>, <domain>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
domain	string	Name of a domain. If this value is `null`, the current domain is assumed.

Returns

Returns an array of strings containing the full pathname of each database operation.

listDBOpsByDBConn

Lists all the database operations associated with the specified database connector.

Request signature

```
listDBOpsByDBConn(<principal>, <dbciName>)
```

Arguments	Type	Description
principal	`Avaki-Principal`	Required authentication information.
dbciName	string	The pathname or qualified name of the database connector. The format for the qualified name is `<domain-name>.<dbconnector-name>`

Returns

Returns an array of strings containing the full pathname of each database operation associated with the database connector.

Chapter 3

Using JDBC drivers

Sybase offers two JDBC (Java Database Connectivity) drivers for use with Avaki EII software:

- The Avaki JDBC driver (supplied with Avaki software)
- JConnect, Sybase's standard JDBC driver (available for download from sybase.com)

Either JDBC driver provides the ability, via JDBC, to:

- Call any data service in the Avaki data catalog
- Call any database operation in the data catalog
- Perform SQL `select` operations against SQL views in the data catalog

Except where noted, the discussions in this chapter apply to both the Avaki JDBC driver and jConnect.

The JDBC drivers allow application programmers to access database data shared in the data catalog using a method already familiar to them. Using this programmatic access you can immediately run further computations or process the data you have retrieved. When a JDBC driver accesses data, it returns a JDBC result set that's immediately available to your program.

This chapter explains how to access the JDBC drivers and how to use them when programming in your environment. It covers these topics:

- "Caching results of database operations" on page 62

- "Prerequisites for using JDBC drivers" on page 65
- "Setting up your application to use a JDBC driver" on page 68
- "JDBC support" on page 75
- "JDBC driver code example" on page 78
- "Using an ODBC driver" on page 80

It is expected that you already have a basic familiarity with programming using JDBC. If you need an introduction to JDBC programming, read the Sun documentation at

http://java.sun.com/j2se/1.4.1/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html

This web page provides a wider view of Sun's JDBC resources:

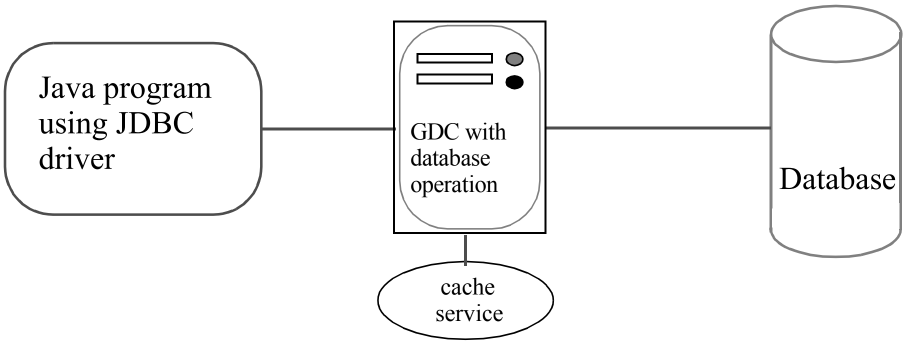
http://java.sun.com/products/jdbc/

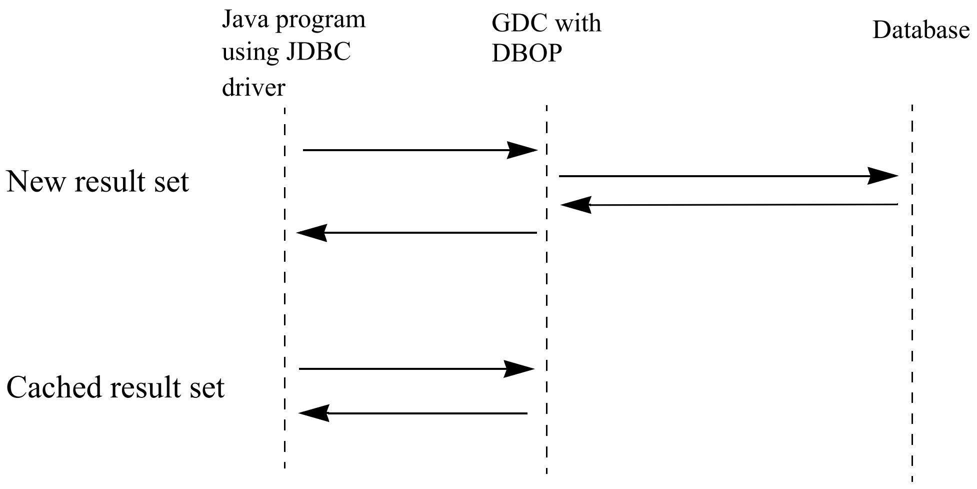
Caching results of database operations

The freshness of result set data obtained using JDBC is affected by the Avaki caching system. If the results have been previously calculated, the freshness of the data is dependent on the coherence window settings for the grid domain controller (GDC) or the database operation. See the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide* for details on how cache services work.

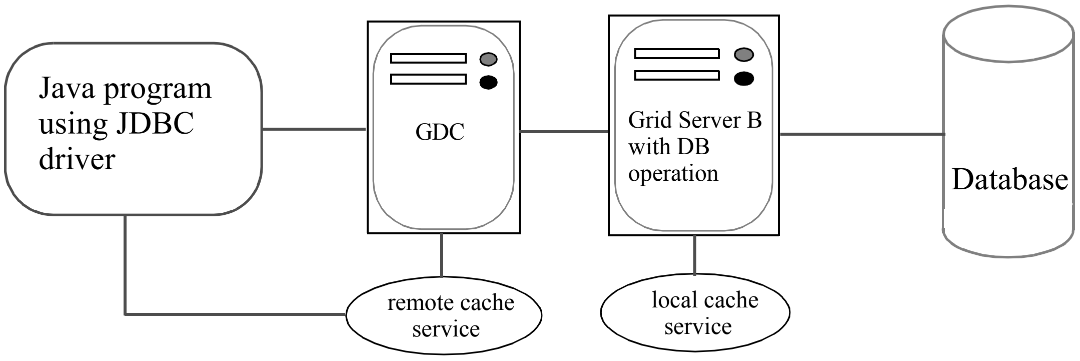
A program using a JDBC driver interfaces with the data grid in slightly different ways, depending on whether you configure a cache service (by setting the externalCacheService property) when connecting the JDBC driver to the data grid.

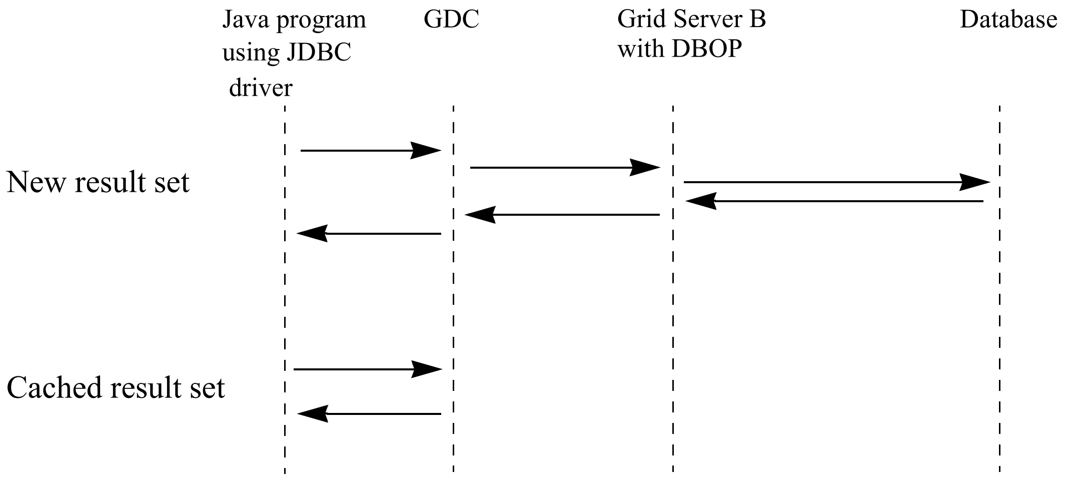
The figure that follows shows how a Java program using a JDBC driver interfaces with the data grid when there is no cache service associated with the Java program. Notice that the Java program benefits from the GDC's cache service.





The figure below shows how a Java program using a JDBC driver interfaces with the data grid when there is a cache service associated with the program. The cache service on the GDC is, in this case, also the cache service associated with the Java program. (This is called the *remote cache* because it's remote from the data source; the cache service on the grid server nearest the data source is called the *local cache*.) When the Java program invokes a database operation, it looks first to its associated cache service on the GDC. If that cache doesn't have the result, the request is passed on to the cache service on the grid server where the database operation resides, Grid Server B. If that cache doesn't have the result either, the query runs on the database and the result is returned to the Java program and stored in both caches.


Java program using JDBC driver
GDC
Grid Server B with DB operation
Database
remote cache service
local cache service



The cache service you associate with your Java program can only be a remote cache. (That is, it cannot be local to the data source—it's ok if the cache is local to your Java program.) If the database operation is configured to use its grid server's local cache, this local caching is entirely separate from the Java program's use of the remote cache. If you don't associate a cache service with your Java program, results will still be cached if the database operation is configured to do so.

If your Java program and database operation use the same grid server, configure caching for the database operation only. There's no reason to associate a cache with the Java program too; the extra level of caching is only helpful when the Java program is remote from the database operation.

Prerequisites for using JDBC drivers

Before using a JDBC driver, ensure that the tasks listed in this section have been completed.

Checking your JRE version

To use a JDBC driver, you must be using the correct version of the Java Runtime Environment (JRE). See the *Sybase Avaki EII Administration Guide* for information on supported versions.

Completing grid setup tasks

The following tasks, which are typically performed by system, grid, or database administrators, must be completed:

For this task...	Look for instructions in...
Install and configure an Avaki domain	*Sybase Avaki EII Administration Guide*
(Optional) Install a JDBC driver for each database you plan to access. Required only if you are using a data service that uses a database operation.	*Sybase Avaki EII Administration Guide*
(Optional) Set up an Avaki database connector for each database you plan to access. Required only if you are using a data service that uses a database operation.	*Data Integration with Sybase Avaki Studio* or *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*
(Optional) Set up at least one of the following to access the database: • One or more Avaki database operations • One or more Avaki SQL views This task is required only if you are using a data service that uses a database operation.	*Data Integration with Sybase Avaki Studio* or *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*

Choosing a JDBC driver

This section helps you decide whether to use jConnect or the Avaki JDBC driver and explains where to find the drivers. It also helps you choose which version of the Avaki JDBC driver to use.

Use these guidelines in choosing a JDBC driver:

- If you're connecting to Avaki from another Sybase product such as ASE, ASA, or Sybase IQ, we recommend using jConnect. Note that jConnect is required to support the CIS feature in ASE.
- If you're not using other Sybase products with Avaki, we recommend using the Avaki JDBC driver, which is generally faster.

jConnect

If you choose to use the jConnect driver, download version 6.0 or later from sybase.com.

Avaki JDBC driver

In <Avaki-install-dir>/examples/lib, Sybase provides two versions of the Avaki JDBC driver JAR file. Both versions include all the support you need to use the Avaki JDBC driver. Choose one of the JAR files:

Avaki_JDBCStandAlone.jar. You can use Avaki_JDBCStandAlone.jar as-is; it has no external dependencies. It includes additional JAR files that allow you to use some third-party software with the Avaki JDBC driver. The third-party JAR files, which are distributed in <Avaki-install-dir>/examples/lib/jdbc-sa/, are:

- antlr.jar
- jax-1_1-fr-qname-class.jar
- js.jar
- log4j.jar
- soap.jar
- xercesImpl.jar
- xml-apis.jar

Included with the third-party JARs is a related file, jdbc-log4j.properties, which you can use to configure log4j logging for the JDBC driver. For information on using log properties files, see the *Sybase Avaki EII Administration Guide*.

Avaki_JDBCStandAlone_Minus3rd.jar. Use Avaki_JDBCStandAlone_Minus3rd.jar only if you find that using the other JAR file, Avaki_JDBCStandalone.jar, results in a conflict involving one of the third-party libraries in Avaki_JDBCStandalone.jar and the version of that library running in your environment.

Avaki_JDBCStandAlone_Minus3rd.jar does not include the third-party JAR files (or the log4j properties file) described above. If you choose this option, you must specify in your classpath the location of a JAR for *every* third-party application you want to use with the JDBC driver, whether they are Avaki-provided JARs or your own versions. See "Configuring your classpath," below, for an example.

Configuring your classpath

For jConnect

Include the following in your classpath if you're using jConnect:

- jconn3.jar

For the Avaki JDBC driver

Include the following in your classpath if you're using the Avaki JDBC driver:

- avakijdbc.properties (optional). If avakijdbc.properties is present, the JDBC driver loads Java system properties from it. A version of this file is provided, packed into the JDBC driver itself. If you need to change system properties, you can either modify the file and repack it into the JAR file, or you can remove it from the JAR file and put it in your classpath. Use the same format for avakijdbc.properties as for Avaki's system.properties file. The property settings in avakijdbc.properties are in effect on the grid server associated with the JDBC driver when the JDBC driver is in use; otherwise the settings in the grid server's own system.properties file prevail. For more information on system properties, including the file format, see the *Sybase Avaki EII Administration Guide*.

- Avaki_JDBCStandAlone.jar or Avaki_JDBCStandAlone_Minus3rd.jar (see "Choosing a JDBC driver," above). You'll find both JAR files in <Avaki-install-dir>/examples/lib.

  If you use Avaki_JDBCStandAlone.jar, you need to include only that JAR file in your classpath. If you use Avaki_JDBCStandAlone_Minus3rd.jar, you must include both your JARs and those supplied by Avaki in your classpath. In this example, we set the classpath on a Windows machine to use a local copy of the Xerces JAR and Avaki-supplied copies of the other third-party JARs:

```
C:\> set CLASSPATH=AVAKI_INSTALL_DIR\examples\lib\
Avaki_JDBCStandAlone_Minus3rd.jar;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\antlr.jar;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\
jax-1_1-ft-qname-class.jar;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\js.jar;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\soap.jar;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\xml-apis;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\log4j.jar;
AVAKI_INSTALL_DIR\examples\lib\jdbc-sa\jdbc-log4j.properties;
E:\dev\lib\xercesImpl.jar
```

Setting up your application to use a JDBC driver

To use a JDBC driver your program must do the following:

1. Load the JDBC driver class.
2. Specify the relevant properties and their values.
3. Provide a connection string (URL) specifying an Avaki grid domain controller. The application passes this string to the JDBC driver, which uses it to connect to the GDC.
4. If your application uses the JDBC driver to execute Avaki database operations or data services, use a stored procedure (`CallableStatement`) pointer to each database operation or data service. The application passes these pointers to the GDC, which uses them to find the database operation or data service that can provide the requested data.

The sections that follow describe the items above in greater detail. The section at the end of this chapter, "JDBC driver code example" on page 78, can serve as a template for setting up your own application to use the Avaki JDBC driver.

Loading the JDBC driver

To use a JDBC driver, load one of the following classes within your application:

For the Avaki JDBC driver: `com.avaki.sql.Driver`

For jConnect: `com.sybase.jdbc3.jdbc.SybDriver`

Connection properties

Your application will use a connect string (described in "Connection strings" on page 71) to acquire a connection to Avaki. In the connect string, you can pass various connection properties to the JDBC driver.

For jConnect

See jConnect documentation for information on connection properties.

For the Avaki JDBC driver

The properties are optional except as noted. Login properties (some of which are required) are listed first, followed by other properties in alphabetical order.

Connection property	**Description**
`user=<user-name>`	Specifies the name of the user account you're using to connect to the data grid. Required.
`password=<password>`	Specifies the user's password. Required.
`auth-service=` `<auth-service-name>`	Specifies the Avaki authentication service that the user account belongs to. Default value: DefaultAuthService
`auth-type=` `<auth-service-type>`	Specifies the type of the authentication service: Grid, Nis, or Ldap. Default value: Grid
`auth-domain=` `<grid-domain-name>`	Specifies the name of the Avaki domain to log in to. Defaults to the local domain of the grid server specified in the connection string.
`adhoc-dbconn=` `[<grid-domain-name>.]` `<dbconn-name>`	Puts the JDBC driver in ad hoc mode, in which it sends database queries using the database connector you specify here. In this mode, Avaki is simply a pass-through to the underlying database. You have no access to SQL views, database operations, or data services when you use adhoc-dbconn.
`domain-dbconn-delimiter=` `<delimiter-character>`	Specifies the character Avaki uses to separate the two elements of the JDBC schema name—the grid domain name and the database connector name. The JDBC schema name, which maps the Avaki namespace to the JDBC namespace, is returned by `DatabaseMetaData.getProce-dures()`. "Bedrock.myDBconn" is an example of a schema name. The delimiter, which defaults to . (period), must be a single character that does not appear in your domain name. **Note:** Some tools don't properly handle a period delimiter in the schema name. If you have trouble viewing or executing Avaki database opera-tions through a third-party database viewer, use this property to specify a different delimiter such as a hyphen.

Connection property	Description
`executionServiceHint=<grid-server-name-or-path>`	By default, the Avaki JDBC driver directs select statements to the query service on the GDC, which load-balances the query execution to any grid server that has been pooled with the GDC. If you want to direct which grid server's execution service will execute your queries, use execution-ServiceHint. The query will be forwarded to that grid server directly for processing.
`externalCacheService=<cache-service-path>`	Required if you want to use an Avaki cache service. Specify the full path in the data catalog for the cache service you want to associate with your Java program. For example, /System/LocalDomain/Servers/myGrid-Server/Services/CacheService.
`FAKE_METADATA={true | false}`	Required if you're using jConnect 5.*x* with Sybase ASE; do not use with other JDBC driver/RDBMS pairs. Set FAKE_METADATA to true for jConnect 5.*x*/ASE. Default value: false
`hideCatalogs={true | false}`	Some packaged applications such as Crystal Reports and BusinessObjects cannot deal with databases that support full three-part names of the form <catalog>.<schema>.<table/view/procedure>. In Avaki, the domain name is mapped to <catalog>, and <schema> maps to one of the following: • The name of a database connector • The name of a deployed metadata model containing mapped tables • The literal "dataservice" (for data services and virtual SQL views generated from them) • The literal "VirtualDb" (for virtual database operations and virtual SQL views generated from them) If you set hideCatalogs to true, the Avaki JDBC driver returns metadata that indicates that Avaki does not support catalogs in its schema. You can then use two-part names of the form <schema>.<object> and Avaki defaults to the current domain as the catalog. Default value: false

Connection property	Description
`queryCacheTTL=<number-of-milliseconds>`	Controls whether the results of SQL queries are cached by the JDBC driver and specifies the number of milliseconds after a SQL query is executed during which the JDBC driver returns cached results if the exact same SQL query is re-executed by the application. (The two queries must be identical. Even minor changes that don't necessarily affect the meaning of the query—such as the addition or removal of spaces, semicolons, and parentheses—cause the JDBC driver to re-execute the query in the virtual database.) Note that this property does not affect stored procedure results, which are not cached. Default value: 0 (do not cache)

You can specify connection properties using a Properties object on the getConnection call or in the URL (connection string) itself.

Avaki supports two different syntaxes for DriverManager.getConnection:

```
DriverManager.getConnection(String url, Properties info)

DriverManager.getConnection(String url, String user,
     String password)
```

The first syntax is shown in "JDBC driver code example" on page 78.

Connection strings

For jConnect

The syntax of the grid server connect string (also called connection URL) for jConnect is

```
jdbc:sybase:Tds:<GDC-machine>:15000[/<database-name>]
[?<first-property>=<value>][&<additional-property>=<value>]+
```

In the syntax above:

<GDC-machine> specifies the DNS name of the machine on which the Avaki domain controller is running.

15000 is the default TDS port used by jConnect. If you need to use a different port, set the port number with the com.sybase.avaki.tdsPort system property. For instructions on setting system properties, see the *Sybase Avaki EII Administration Guide*.

<database-name> specifies the database to connect to. Optional.

<first-property>=<value> and <additional-property>=<value> specify connection properties and their values. Notice that <first-property> is preceded by ? (question mark) and each <additional-property> is preceded by & (ampersand). Optional.

For example:

```
jdbc:sybase:Tds:myMachine.sybase.com:15000

jdbc:sybase:Tds:myMachine.sybase.com:15000/myDatabase?user=Admini
strator&password=SY4evr

Connection conn =
DriverManager.getConnection("jdbc:sybase:Tds:localhost:15000",
"Administrator", "SY4evr");
```

For the Avaki JDBC driver

The syntax of the grid server connection string (also called connection URL) for the Avaki driver is

```
jdbc:avakisql:@<GDC-machine>[:<grid-port>]
     [;<property-name>=<value>]+
```

In the syntax above:

<GDC-machine> specifies the DNS name of the machine on which the Avaki domain controller is running.

<grid-port> specifies the connect (JNDI) port of the GDC. Optional; defaults to 3099.

<property-name>=<value> specifies a connection property (from "Connection properties" on page 68) and its value. Insert a semicolon before each property=value pair. Optional.

For example:

```
jdbc:avakisql:@myMachine.sybase.com

jdbc:avakisql:@myMachine.sybase.com:1017;user=Administrator;passw
ord=SY4evr;auth-domain=Burlington

Connection conn =
DriverManager.getConnection("jdbc:avakisql:localhost:1017;adhoc-d
bconn=my-db-conn", "Administrator", "SY4evr");
```

Data services

You can use JDBC's `CallableStatement` syntax to call data services via the JDBC drivers. Data services are exposed as callable procedures. Results are returned as a JDBC result set.

The syntax of the data service pointer is:

```
CallableStatement stmt = conn.prepareCall(
     "{[? =][<grid-domain-name>].dataservice.
     <data-service-name> [...]}")
```

`<grid-domain-name>` specifies the name of the Avaki domain in which the data service is located. This is optional. Specify this if the data service you require is not in the same grid domain specified in the connection string.

`<data-service-name>` specifies the name of the Avaki data service that provides the data your application needs.

When you call data services, be sure to pass any parameters in the order in which they appear in the data service descriptor, in Avaki Studio, and in the web UI. To display a web UI page that includes a properly ordered list of the parameters for a data service, log in to the web UI and navigate to the View Data Services screen:

Home > Data source management > View data services

Then click the **View/Edit** link for the data service of your choice.

Database operations

You can use JDBC's `CallableStatement` syntax to call database operations via a JDBC driver. Database operations are exposed as callable procedures. Results are returned as a JDBC result set.

The syntax of the database operation pointer is:

```
CallableStatement stmt = conn.prepareCall(
     "{[? =][<grid-domain-name>].<db-connector-name>.
     <dbop-name> [...]}")
```

In the syntax above:

`? =` is used if the call returns output parameters.

`<grid-domain-name>` specifies the name of the Avaki domain in which the database operation is located. This is optional. Include this if the database operation you require is not in the same grid domain specified in the connection string.

`<db-connector-name>` specifies the name of the Avaki database connector that connects to your database. If you're calling a virtual database operation, use the value "VirtualDb".

`<dbop-name>` specifies the name of the Avaki database operation (or virtual database operation) that provides the data your application needs.

Note You cannot call Avaki database operations with the Avaki JDBC driver when the driver is in ad-hoc mode (using the adhoc-dbconn property).

Pass-through ad-hoc queries

In ad-hoc mode, the Avaki JDBC driver passes all JDBC operations directly to the RDBMS JDBC driver that is running in the grid server. In this mode you can perform almost all JDBC operations that the RDBMS driver will support. Using a CallableStatement in ad-hoc mode results in a call to a stored procedure or function in the back-end RDBMS, and not to a grid operation.

Note Do not specify ad-hoc mode if you want to query SQL views in the Avaki virtual database. A regular JDBC connection gives you full access to SQL views as well as database operations and stored procedures.

An example of ad-hoc usage:

```
Connection conn =
     DriverManager.getConnection("jdbc:avakisql:localhost;
     adhoc-dbconn=my-dbci",
     "Administrator", "Administrator");

Statement s = conn.createStatement();
s.execute("select * from emp");
ResultSet rs = s.getResultSet();

PreparedStatement ps = conn.prepareStatement("select * from
     emp where ename = ?");
ps.setString(1, "FRED");
ps.execute();
ResultSet rs = ps.getResultSet();
```

JDBC support

This section briefly describes the scope of JDBC support in Avaki.

DatabaseMetaData

This section provides details on how Avaki uses the java.sql.DatabaseMetaData interface. The following methods get information about the Avaki domain and its data resources:

- `getCatalogs()` returns a single row containing the name of the current Avaki domain. (That is, the Avaki domain name maps to "catalog" in JDBC.) If there are interconnected domains, `getCatalogs()` returns a row for each interconnected domain as well.
- `getCatalogTerm()` returns "Domain"
- `getSchemaTerm` returns "SCHEMA"
- `getSchemas()` returns the following:
  - One row that corresponds to each database connector
  - One row containing "dataservice", which is the schema for all data services
  - "VirtualDb", which is the schema for virtual database operations and SQL views generated from them
  - One row that corresponds to each deployed metadata model. Model names take the form [<catalog>.]<schema>, so if a model is deployed as foo.bar, the schema name returned will be "bar." For a model deployed as foo, the schema name returned will be "foo." (The catalog name is not returned.) For more on metadata models, see *Data Integration with Sybase Avaki Studio*.

As long as table/view names are unique, you don't need to qualify them with catalog and schema names in `select` statements.

The table type for provisioned SQL views is "TABLE"; for virtual SQL views (that is, SQL views generated from database operations or data services) it is "View".

Supported result set types

Avaki supports the following result set types:

`<ResultSetType>` only `ResultSet.TYPE_FORWARD_ONLY` is supported. Avaki does not support scrollable result sets.

`<ResultSetConcurrency>` only `ResultSet.CONCUR_READ_ONLY` is supported. Avaki does not support updatable result sets.

`<ResultSetHoldability>` only `ResultSet.HOLD_CURSORS_OVER_COMMIT` is supported. Avaki supports only holdable cursors.

Error messages

If your program attempts a JDBC operation not supported by Avaki, Avaki generates an exception; the stack trace of the exception will contain an error message explaining the problem. For example, if you specify the `ResultSetType` in `prepareCall` as `ResultSet.TYPE_SCROLL_SENSITIVE`, which Avaki does not support, you will see:

```
Execution of Avaki DBO failed.

com.avaki.core.sql.SQLKeyedException: Unsupported
ResultSet type indicated.
```

Data types

Access to a database through the Avaki JDBC driver supports all the basic JDBC data types. These are:

```
CHAR, VARCHAR, and LONGVARCHAR
BINARY, VARBINARY, and LONGVARBINARY
BIT
TINYINT
SMALLINT
INTEGER
BIGINT
REAL
DOUBLE
FLOAT
DECIMAL and NUMERIC
DATE, TIME, and TIMESTAMP
```

In addition, the following advanced data types are supported: `BLOB` and `CLOB`. Because these types are stored in memory as a single atomic unit and not streamed, their maximum size depends on your JVM heap space size.

The following advanced types are not supported:

```
ARRAY
DISTINCT
STRUCT
REF
JAVA_OBJECT
```

For a full description of JDBC data types, read the following Sun specification documentation. This material also explains how JDBC types are mapped on to the underlying database types.

http://java.sun.com/j2se/1.4.1/docs/guide/jdbc/getstart/mapping.html#996857

Batch mode

Avaki supports JDBC batch mode. To use JDBC batch mode, you first create a callable statement, then set parameters for each separate call on that statement and call `addBatch` between sets. Next, call `executeBatch`, and all of the statements that you batched up will get sent out. The following code provides an example:

```
CallableStatement stmt = connection.prepareCall("{call
dbci.dbo(?, ?)}");

    stmt.setInt(1, 1);
    stmt.setString(2, "Tom");
    stmt.addBatch();

    stmt.setInt(1, 2);
    stmt.setString(2, "Dick");
    stmt.addBatch();

    stmt.setInt(1, 3);
    stmt.setString(2, "Harry");
    stmt.addBatch();

    stmt.executeBatch();
```

If this callable statement inserts the given data in a table, it creates the following table in the database:

Number	Name
1	Tom
2	Dick
3	Harry

JDBC driver code example

The code in this section is a functioning example of how to use the Avaki JDBC driver. You can use it as a model to build your own applications.

To run the sample code, you need a database operation that selects rows from a database. If you don't have such a database operation, see the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide* or *Data Integration with Sybase Avaki Studio* for instructions on setting up database operations.

Follow these steps to run the example:

Step 1 Cut and paste the sample code into a source file named AvakiJDBCExample.java.

Step 2 Edit the file and change the connection properties (starting at the first pointer) to match your configuration. Specifically, change the host name in the string URL and in the cache service path; also change the user, password, and, if needed, the authentication service parameters.

Step 3 Change the qualified name of the database operation (at the second pointer) to that of a database operation in your system.

Step 4 Compile the program:

```
javac AvakiJDBCExample.java
```

Step 5 Set your CLASSPATH to include javaki_JDBCStandAlone.jar (located in the <Avaki-install-dir>\examples\lib directory). In Windows:

```
set CLASSPATH=c:\ADG\examples\lib\javaki_JDBCStandAlone.jar;.
```

Step 6 Run the program:

```
java AvakiJDBCExample
```

The program should display the number of rows returned by your database operation.

Here's the code example:

```
import java.sql.*;
import java.util.Properties;


public class AvakiJDBCExample {

  public static void main(String args[]) {
    Connection con = null;
    CallableStatement cs;
    ResultSet rs;
    int rows;

// Attempt to load Avaki JDBC driver.
    try {
      String driver = "com.avaki.sql.Driver";

      Class.forName(driver).newInstance();
    }
    catch( Exception e ) {
      System.out.println("Failed to load Avaki JDBC driver.");
      e.printStackTrace();
      return;
    }

//  Attempt connection to Avaki Grid
    try {
      Properties avakiprops = new Properties();
      String url = "jdbc:avakisql:@gridserver1.avaki.com:3099";

      avakiprops.setProperty("user", "jdoe");
      avakiprops.setProperty("password", "jdoepassword");
      avakiprops.setProperty("auth-type", "Grid");
      avakiprops.setProperty("auth-service", "DefaultAuthService");
      avakiprops.setProperty("auth-domain", "mygrid");
      avakiprops.setProperty("externalCacheService",
      "/System/LocalDomain/Servers/gridserver2.company.com/
      Services/CacheService");
      con = DriverManager.getConnection(url, avakiprops);
    }
    catch ( SQLException e) {
      System.out.println("Connection to Avaki Grid failed.");
      e.printStackTrace();
      return;
    }

//  Execute Avaki Database Operation via JDBC Driver
    try {
      cs = con.prepareCall("{call mygrid.mydbconn.mydbo(?)}",
ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY,
ResultSet.HOLD_CURSORS_OVER_COMMIT);
      cs.setString(1, "SCOTT");
      rs = cs.executeQuery();
    }
    catch( SQLException e ) {
      System.out.println("Excecution of Avaki DBO failed.");
      e.printStackTrace();
      return;
    }

//  Perform simple action (count rows returned in ResultSet rs)
    try {
      int count = 0;

      while (rs.next()) { count++; }
      System.out.println(count + " rows returned.");
    }
    catch( SQLException e ) {
      System.out.println("ResultSet traversal failed.");
      e.printStackTrace();
      return;
    }

//  Close JDBC connection to Avaki Grid
    if ( con != null ) {
      try { con.close(); }
      catch ( SQLException e ) {
        System.out.println("Close of JDBC Connection Failed");
        e.printStackTrace();
      }
    }
  }
}
```

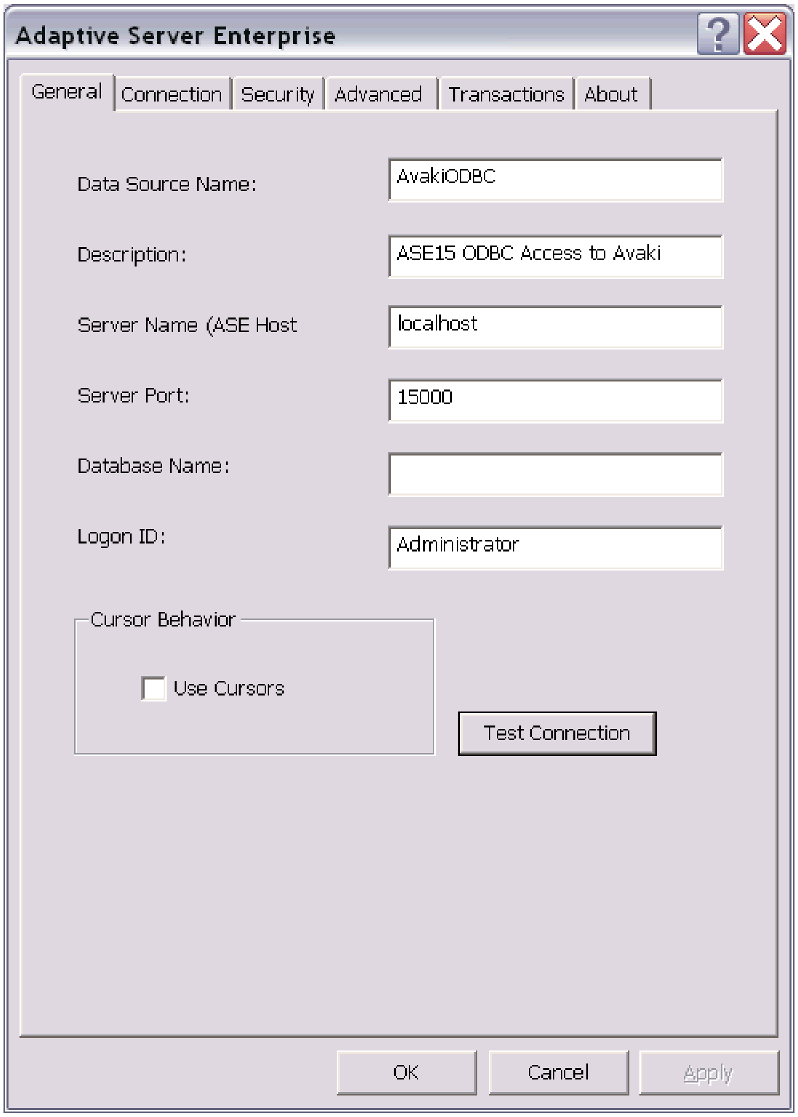
Using an ODBC driver

An ODBC (Open DataBase Connectivity) driver allows Avaki to communicate with Windows database applications. Avaki is compatible with and has been tested with the Sybase ODBC driver included with ASE 15.0. Instructions on setting up this ODBC driver are included in the documentation for Sybase ASE.

When you set up an application to use an ODBC driver to access Avaki data, include the following information to identify your data source:

- The name of the Avaki object providing the data—the data service, database operation, SQL view, or other object
- The name of the grid server machine on which the Avaki data source resides
- The port number for connecting to Avaki: 15000

Here’s a sample Windows configuration screen for the Sybase ODBC driver:


Adaptive Server Enterprise

General | Connection | Security | Advanced | Transactions | About

Data Source Name:	AvakiODBC
Description:	ASE15 ODBC Access to Avaki
Server Name (ASE Host	localhost
Server Port:	15000
Database Name:	
Logon ID:	Administrator

Cursor Behavior

☐ Use Cursors

Naming Avaki objects that will be accessed by BusinessObjects. Don’t use underscore characters in the names of any Avaki provisioned tables, database operations or data services whose results you plan to access using the BusinessObjects query and analysis tool. BusinessObjects is not able to browse the columns of Avaki objects whose names include underscores.

Glossary

Terms printed in *italics* are defined in the glossary.

access control list
(ACL) A list, for a given file, directory, or other Avaki object, of permissions—read, write, execute, delete, and owner—that control which users and groups can view, modify, invoke, and remove the object, and edit the object's ACL.

ACL
See *access control list*.

ad-hoc query
A mechanism that lets you directly query a database in SQL. The query must run through an existing Avaki *database connector*. You can run an ad-hoc query using either the CLI or a *JDBC driver*. Ad-hoc queries can be thought of as single-use *database operations*.

attribute
A property of an *Avaki directory*, file, *service*, or other object. Each attribute has a name, a type (string, integer, float, date, time, or timestamp) and a value. System attributes are read-only; you can change the values of other attributes. You can also create new attributes and add them to objects as needed.

authentication service
A *service* associated with an *Avaki domain* that authenticates an Avaki user's identity and provides security credentials each time the user logs in. Avaki can be configured to use third-party directory services as authentication services for login; for user accounts created directly in the Avaki domain, Avaki uses its own default authentication service.

Avaki directory
Avaki software creates a single, unified namespace that is accessible (subject to Avaki *access control lists*) to all users in the *Avaki domain*. The namespace, called the *data catalog*, is arranged as a hierarchy of Avaki directories (folders). The catalog directory structure is stored by the domain's grid servers and its GDC, while the physical files remain in their original locations in your local file systems. When you work with directories, it's important to distinguish between Avaki directories, which are part of the data catalog, and local directories, which reside in your local file system.

Avaki domain
The basic administrative unit of the Avaki EII system. An Avaki domain consists, at a minimum, of one *grid domain controller* and may also include one or more *grid servers*, *share servers*, *proxy servers*, *data grid access servers*, and *command clients*. See also *domain name*.

Avaki group
A set of users who have the same permissions on one or more Avaki objects. You can use the group name in place of a user name when you set permissions or create *access control lists*.

Avaki installation directory
The directory in your local file system where Avaki software is installed. This is not a *data catalog* directory.

Avaki share
(Also shared directory.) A pointer in the Avaki *data catalog* to a directory or file in the underlying local file system. When you browse the data catalog, Avaki shares look like—and can be accessed like—other Avaki directories. Contrast with *CIFS share.*

Avaki server
A *service* that starts, stops, and monitors other Avaki services on a particular computer. Every server is part of an *Avaki domain*. A server is permanently attached to the computer where it is started. There are several types of server: *data grid access servers*, *grid domain controllers*, *grid servers*, *share servers*, and *proxy servers*.

Avaki Studio
A graphical, metadata-based data integration tool that lets you

- Build data flows by dragging and dropping input sources, operators, and output targets. You can deploy your data flows as Avaki *data services*.
- Import or create *metadata models* and apply them to Avaki objects or use them to build new data services.

In addition, you can use Studio to perform provisioning tasks (creating *database connectors*, *database operations*, *virtual database operations*, and *SQL views*), manipulate *categories*, and edit *ad-hoc queries* and *attributes*.

cache service

(Formerly proxy cache service.) A staging service that stores copies of files, *database operation* results, and *data service* results. Caching improves retrieval performance. To ensure that an object is stored in the cache, you can *pin* a file or directory in the data catalog, or schedule a database operation or data service. A cache service can provide remote caching, local caching, or both. The freshness of cached data is controlled by a data expiration interval that determines how long cached data is considered valid and by a cache coherence window that tells the cache service how often to check whether cached data is still valid. If cached data is too old to satisfy a new request (or is not stored in this cache), the cache service does one of the following:

- If the database operation or data service that produced the data is local to this cache service, the cache service triggers execution of the database operation or data service.
- If the database operation or data service that produced the data is remote from this cache service, this cache service requests the data from the data source's local cache service.

A cache service can be associated with a *data grid access server*, a *grid server*, or a local user in a CLI session. See also *local cache*, *remote cache*, *on-demand caching*, and *scheduled caching*.

category

A mechanism for classifying and organizing the contents of the *data catalog*. Like *Avaki directories*, categories serve as containers for objects in the data catalog. Anything in the data catalog—views, data services, shared files, even Avaki directories themselves—can be assigned to a category. Categories are hierarchical, they have attributes, and Avaki *access control lists* regulate access to them.

CIFS client

A machine that mounts files or directories from the Avaki *data catalog* by connecting to a *CIFS share* through an Avaki *data grid access server*. A CIFS client need not have Avaki software installed. (CIFS—Common Internet File System—is a file-sharing protocol based on the file system implemented by Windows.)

CIFS share

A directory or file that has been exported (shared) from the Avaki *data catalog*. A CIFS share can be mapped into a Windows file system like a network drive. When you browse the Windows file system, CIFS shares look like—and can be accessed like—other files and directories. CIFS shares are created through a *data grid access server*. Contrast with *Avaki share*.

client

Avaki supports several types of client: *Avaki Studio*, *CIFS clients*, *command clients*, JDBC/ODBC clients, *NFS clients*, *web clients, and WS clients*.

command client

A machine that can issue Avaki commands but does not contribute resources to the *Avaki domain*.

connect port

The connect port on a *grid domain controller*, *grid server*, *data grid access server*, *proxy server*, or *share server* accesses the JNDI naming service or RMI registry for the underlying application server. The connect port is one of many ports that a GDC or server uses to communicate with other Avaki objects. You must supply the connect port number of a target grid server or GDC whenever you connect a new object (another server, a copy of Avaki Studio, or a *command client*, for example) to an *Avaki domain*. When you *interconnect* two Avaki domains, you must supply each domain's connect port number to the other one.

data catalog

A hierarchical structure similar to a file system that encompasses all objects in an *Avaki domain*. The data catalog contains *Avaki directories* and files, *Avaki shares*, *Avaki servers*, *SQL views*, *database operations* and *data services*, and other objects.

data grid access server

(DGAS) An *Avaki server* that makes *Avaki directories* and their contents available to *CIFS clients* and *NFS clients*.

data service

An operation that transforms data obtained from sources in the *data catalog*. Input data can come from any number of sources, including:

- other data services
- data catalog files (which can be *generated views*)
- Avaki *database operations* (which in turn extract the data from relational databases)
- HTTP requests
- Web service invocations

You can generate the code that manipulates the data by creating a *view model* in *Avaki Studio*, or by writing a custom *data service plug-in* using Java, JavaScript, or XSLT. Data service output can be in rowset or XML format. Data services are run by the *execution services* on *grid servers*, they can be scheduled, and their results can be cached.

data service plug-in
The logic for a *data service*, written in Java, JavaScript, or XSLT. Data service plug-ins are modular—you can use the same plug-in for multiple data services. *Avaki Studio* creates data services and plug-ins simultaneously, so if you use Avaki Studio to create data services, you don't have to worry about plug-ins. You can also use the Avaki Plug-in Wizard to create data service plug-ins.

database connector
A mechanism that enables one or more *database operations, SQL views*, or *ad-hoc queries* to connect to a relational database.

database operation
(DBOP) A mechanism that can

- extract data from a relational database and deliver it on demand to a *view generator* or a *data service*, or
- modify data in a relational database.

A database operation can be a SQL statement or a stored procedure call.

dependency
A relationship in which an Avaki object requires input from other Avaki objects. A *data service* might require input from one or more *database operations* or from other data services. A *view generator* might depend on a database operation for input. A database operation can serve as an input source for one or more data services or view generators. Generated *SQL views* depend on database operations, virtual database operations, or data services. You can use *Avaki Studio*, the web UI, or the CLI to list input and output dependencies for any data service, database operation, or view.

DGAS
See *data grid access server*.

distributed transaction
A set of related operations (typically SQL operations such as SELECT, INSERT, UPDATE, DELETE, and CALL) that

- involve one or more databases, and
- might lead to unwanted results (such as leaving participating databases in an inconsistent state or producing inconsistent reads) if some of the operations complete and others do not, and therefore
- must all be executed at once, as a single transaction.

The individual operations that make up a distributed transaction are performed by *database operations* that use *database connectors* configured with XA-capable *JDBC drivers*; all the database operations are executed, using the two-phase commit protocol, by a specially configured *data service*. The two-phase commit protocol is designed to ensure that the participating databases will be left in a consistent state—that is, that all the operations in the distributed transaction will be completed, or none of them will.

domain name
A unique alphanumeric identifier for an *Avaki domain*. The domain name is assigned by the Avaki administrator when the Avaki domain is initialized. The domain name has a maximum length of 30 characters.

enterprise information integration
(EII) A software system that

- enables applications and users to access, without replication, both raw and integrated data from multiple heterogeneous distributed data sources while hiding the complexity of the data sources, and
- provides tools enabling users and data owners to further integrate and transform data.

exclusion
See *schedule exclusion*.

execution service
Execution services execute *data services*. There is an execution service on every *grid server*, and you can configure a pool of execution services for load-sharing. When a pool is in place, a data service can be run by any execution service in its grid server's pool.

failover
The transition of control from a failing or unreachable primary *grid domain controller* to a secondary grid domain controller.

federated data access
A scheme that allows independently controlled elements to be shared into a single namespace. Files, user accounts, and other objects maintain their separate identities and remain under the control of their owners, but—subject to access controls—the objects can be accessed, managed, and viewed as if they were part of a single system.

GDC
See *grid domain controller*.

generated view
A file created by a *view generator;* it may contain data obtained from a *database operation*, a *data service*, a file, or an HTTP source. Like other files, generated views exist in a local file system and are shared into the *data catalog*.

grid
A heterogeneous group of networked resources that appears and functions as one operating environment. A data grid like the Avaki Enterprise Information Integration (EII) system provides secure, shared access to data.

grid directory
See *Avaki directory*.

grid domain
See *Avaki domain*.

grid domain controller
(GDC) The first server in an *Avaki domain* is the grid domain controller. The GDC maintains a portion of the Avaki domain's namespace and provides authentication services. It can also run Avaki commands, share data, and monitor other servers. (That is, the GDC functions as a *grid server.*) If the domain is configured for *failover*, it has both a primary GDC and a secondary GDC; the secondary is updated at regular intervals and takes over management of the domain if the primary fails. Any Avaki shares managed by the primary are read-only on the secondary.

grid server
An *Avaki server* that maintains a portion of the *Avaki domain*'s namespace, runs Avaki services such as shares, execution services, caches, and searches, and allows you to run Avaki's web UI and execute Avaki commands.

group
See *Avaki group*.

hard link
Provides an alternate name for an item in the *data catalog*. Changes to the object's other names have no effect on the hard link: you can move or change a file's original name and the hard link will still know where to find the file. To delete a hard-linked object, you must remove its original name. Contrast with *soft link*.

interconnect
To create a unidirectional link from one *Avaki domain* to another. Interconnecting lets an Avaki domain make its *data catalog* visible to users in another domain (subject to Avaki access controls).

JDBC driver
JDBC (Java Database Connectivity) drivers allows application programmers to access database data shared in the *data catalog*. When a JDBC driver accesses data, it returns a JDBC result set that's immediately available to your program. JDBC drivers can:

- Call any *data service* in the data catalog
- Call any *database operation* in the data catalog
- Perform SQL `select` operations against *SQL views* in the data catalog

Sybase offers three JDBC drivers for use with Avaki EII software:

- The included Avaki JDBC driver
- jConnect, Sybase's standard JDBC driver
- An XA-capable driver for use with *database connectors* that support *distributed transactions*

link
See *hard link*, *soft link*.

local cache
A *cache service* that runs on the same *grid server* as a *database operation* or a *data service* that generates cachable data. The local cache stores results produced by local database operations and data services so they don't have to execute for every new request. See also *remote cache*.

metadata model
A construct in *Avaki Studio* that expresses a schema by defining a set of tables. A table in a metadata model can be mapped (linked) to an Avaki object such as a *data service* or a *database operation*, or to a table in a relational database. The mapping lets you address each mapped object by the name of the corresponding table in the metadata model. You can also derive a *view model* schema from a metadata model. When you do this, you ensure that the results of any data service deployed from the view model will conform to the metadata model's schema.

NFS client
A machine that mounts the Avaki *data catalog* (or a portion of it) as a directory by connecting to an Avaki *data grid access server*. An NFS client need not have Avaki software installed. (NFS—Network File System—lets you add file systems located on a remote computer to the directory structure on your own computer.)

ODBC

ODBC (Open DataBase Connectivity) is an API for databases on Windows. An ODBC driver (such as the the Sybase Organic ODBC driver included with Sybase ASE) allows Avaki to communicate with Windows database applications.

on-demand caching

A scheme by which an object is cached only if it's used—for example, results are cached when a *database operation* or a *data service* is executed, or a file is cached when a user or application reads it. On-demand caching uses a fixed expiration interval to determine data freshness. On-demand caching is suitable for objects that are rarely accessed or that change at irregular intervals. Contrast with *scheduled caching*.

pin

To mark an *Avaki directory* or file for *scheduled caching*. See also *cache service*.

plug-in

See *data service plug-in*.

primary GDC

See *grid domain controller*.

proxy server

An *Avaki server* that allows *Avaki domains* on opposite sides of a firewall or a Network Address Translator (NAT) to communicate with one another.

queries

See *ad-hoc query*.

query engine

An Avaki *service* that executes SQL queries against the *SQL views* (tables) that make up the Avaki *virtual database*. A query engine analyzes queries, pushes as much of the work as possible down to the underlying relational database (if there is one), and performs the remaining operations (such as joins across tables from different databases) itself. There is a query engine on each *grid server*.

remote cache

A *cache service* that runs on a grid server that is remote from an Avaki service (a *database operation* or a *data service*) that generates cachable data. The remote cache stores results produced by distant services so the results don't have to be fetched over the network to satisfy every new request. Users and applications that access remote data through the cache may have access to cached copies even when the remote data source is unavailable. See also *local cache*.

scheduled caching
A scheme by which an object is cached according to a schedule that you create. The schedule specifies when the object is first cached and how often (or following what trigger event, such as a change to a file) the cache is refreshed. If the object is a *data service* or a *database operation*, the schedule runs it to put fresh results in the cache. Scheduled caching, which overrides other types of caching, is suitable for objects that are updated frequently or on a regular basis. Contrast with *on-demand caching*.

schedule exclusion
A named period of time during which scheduled activities can be prevented from running. You can apply an exclusion to as many schedules as you want. Scheduled activities include refreshing *Avaki shares* and imported user accounts, and caching files, directories, and the results of *database operations*, *data services*, and *generated views*.

secondary GDC
See *grid domain controller*.

service
An Avaki object that performs a function in the domain (stores data or authenticates users, for example). Services provided in Avaki software include *Avaki directories*, *Avaki shares*, *Avaki servers*, *authentication services*, *execution services*, and user accounts.

share
A point of connection between the Avaki *data catalog* and a native file system or file system tool. Avaki supports two kinds of shares: *Avaki shares* and *CIFS shares*.

share server
An *Avaki server* whose only task is to manage *Avaki shares*—local directories that are exported (shared) into the *data catalog*. (Grid servers can also manage shares.)

shared directory
See *Avaki share*.

soft link
A pointer to a particular location (name) in the Avaki *data catalog*. If the object at that location is moved, deleted, or renamed, the soft link leads nowhere. Soft links can be created only in the CLI. Contrast with *hard link*.

SQL view
A virtual table—a *data catalog* entry that represents a table in a relational database, a *database operation*, or a *data service*. SQL views can be created in three ways:

- Provisioned directly from a table in an underlying database
- Generated from a database operation or data service
- Mapped from a database table, a database operation, or a data service, using the *Avaki Studio* metadata model editor

Every SQL view is part of the Avaki *virtual database*. SQL views are treated as relational tables by the Avaki *query engine*. SQL view data can be accessed using standard SQL statements by connecting to Avaki with ODBC or JDBC, or via an Avaki *virtual database operation*.

update notification

A message issued when a *generated view* is updated. A view that receives data from another view can be configured to regenerate itself (using the new data) upon receipt of an update notification.

view generator

A mechanism that does one of the following: extracts data from a file or an HTTP source, obtains data from an Avaki *data service*, or uses an Avaki *database operation* to extract data from a relational database. The view generator can display the data, perform an XSLT transform, save the data as a *generated view* file, and/or update a database. Contrast with *data service*.

view model

The graphical representation of a data flow that you can build in *Avaki Studio*. A view model typically includes one or more input sources (such as *database operations* or *data services*), one or more operations to combine or transform the data, and an output target. When you deploy a view model, it becomes an Avaki data service.

virtual database

The set of all *SQL views* in an *Avaki domain*, including those provisioned from external databases and those generated from *data services* and *database operations*. You can execute SQL queries on the SQL views in the virtual database as if they were tables in a single database.

virtual database operation

A *database operation* whose source database is the Avaki *virtual database* itself. Use virtual database operations if you want to encapsulate and reuse SQL SELECT queries against *SQL views* (provisioned or generated).

web services client

See *WS client*.

WS client

(Also web services client.) A tool or a piece of code that is part of a customer application and that makes SOAP calls to web services on an Avaki grid server. The SOAP calls can request data from the Avaki *data catalog*, from a *database operation*, or from a *data service*.

Master Index

In electronic copies of this book, the index links to other books in the documentation set work only as long as the PDF files are stored in the same directory.

Key

AD: *Administration Guide*
API: *API Guide*
C: *Command Reference*
O: *Overture*
P: *Provisioning & Advanced Data Integration Guide*
S: *Data Integration with Avaki Studio*