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

# Sybase Avaki EII Overture

Release 7.0

August 24, 2006

DOCUMENT ID: DC00572-01-0700-01

LAST REVISED: August 2006

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 Overture*
Written by Beth Thoenen, Cheryl Magadieu, Stephanos Bacon, and Linda Thorsen

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

# Table of contents

# Preface

Just as an operatic overture introduces themes that are repeated later in the opera, this manual introduces ideas that you will encounter in your work with Sybase Avaki EII software. The *Sybase Avaki EII Overture* explains concepts you'll need to understand to use, install, or administer an Avaki domain. It also describes the components and structure of an Avaki domain, including the roles of the various Avaki servers, the purpose of Avaki Studio, and the contents of the Avaki data catalog. Read the *Overture* before using Avaki software or reading other books in the documentation set.

This book is intended for everyone who uses or administers an Avaki system.

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

## Organization

The *Sybase Avaki EII Overture* is organized as follows:

| | |
|---|---|
| Chapter 1<br>Introduction to Avaki | Provides an overview of the purpose and structure of the Avaki solution. |
| Chapter 2<br>Database tools | Introduces tools for provisioning and integrating database data. |
| Chapter 3<br>Browsing the data catalog | Describes the data catalog, discusses the names used to address objects in the catalog, and summarizes the contents and purpose of the default top-level Avaki directories. |
| Chapter 4<br>Authentication and access control in Avaki domains | Explains how login and permissions work. |
| Chapter 5<br>Caching | Describes how Avaki cache services work. |
| 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 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.

# 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

# Introduction to Avaki

Sybase Avaki Enterprise Information Integration (EII) software simplifies provisioning, access, and integration of distributed data—for one group or across the extended enterprise. You can integrate relational data, XML documents, files, and application data across departments, locations, and companies and allow access to authorized users via a number of protocols and interfaces including transparent file access, ODBC, JDBC and SOAP.

Not a replication solution, Avaki EII links existing data into a *data catalog*, where—subject to fine-grained access controls—it can be accessed from anywhere in the enterprise.

This chapter covers the following topics:

## The distributed data challenge

Today's companies need data to execute cross-functional or cross-location workflows, accelerate product innovation and development, and analyze business trends to respond swiftly to changing market conditions. As a result they spend too much time and money to accomplish one very simple goal: give users, developers, and applications secure, real-time access to heterogeneous distributed data. For global, distributed organizations, the challenge—and corresponding costs—are magnified by mergers, acquisitions, and reorganizations, heterogeneous technology infrastructures, and the sheer number and diversity of data sources.

Sybase Avaki EII software helps companies provide more data to more people faster and at lower cost. With Avaki, IT managers can reduce the backlog of data requests while implementing a simple, secure, and uniform approach to provisioning, access, and integration of distributed data. And they can do so in an incremental, nondisruptive way that lets everyone involved work the way they want to work.

Avaki is a different kind of information integration software. Other integration solutions might address only a single group's need for data, yet require significant time and effort to deploy and maintain. Avaki gives your organization a single, unified framework—a *data grid,* supporting a data services layer that provides a set of reusable services for accessing data across the organization, without regard to what the data is, where it resides, or how to access it.

Owners of data sources—be they packaged applications, databases or file systems—can provision data and control access to it by setting the appropriate permissions. This provisioned data is accessible and re-usable as named services. Data architects, application developers or anyone with the appropriate permissions can then use a number of techniques and technologies to register services that combine and transform the provisioned data. End users and consuming applications can access this data using standard interfaces and protocols.

For many organizations, Avaki offers breakthrough reductions in cost and time-to-solution compared with alternatives such as centralized information integration solutions. A solution with a federated approach that leaves data in place, under control of local owners, Avaki can be deployed easily without disrupting users, applications, or data management procedures. Avaki creates a linked data layer rather than copying source data. Its pluggable, distributed architecture lets partners create a secure, streamlined infrastructure for sharing only the data they want to share—a major advantage over centrally-controlled solutions.

Avaki can provide budget relief for projects such as delivering data to business intelligence applications, corporate dashboards, or portals, creating a single view of the customer, or making distributed data available for analytical applications in product design, drug development, or risk management. Once projects are complete it can reduce cost of ownership while enabling the work to be reused elsewhere in the organization. This chapter provides a conceptual overview of Avaki software.

## Data provisioning and integration

Data provisioning—the process of making data available in an orderly and secure way to users, application developers, and applications that need it—is a significant challenge for large, distributed organizations. With many widely varying demands for data, geographically distributed users and data sources, production systems that must be insulated from uncontrolled access, and concerns about intellectual property and confidential data, careful data provisioning is more important and more difficult than ever before.

If everyone needed data in the same format from a single data source, and that format happened to be the way the data is currently stored, there would be no data integration challenge. But the world is not that simple. Some applications expect relational data. Others need data in XML form. Still others need to aggregate sales data across multiple departments, or integrate data from different systems to obtain a single view of the customer. Data of multiple types must be combined to provide a result.

All this poses significant challenges to application developers, who must spend time writing code to access and transform data, rather than writing business logic. Developers also need to know where the data resides, and changes in the location typically break the application. Avaki EII shields developers from this issue, as only the service definitions have to be changed when data moves—not the application code.

While many software vendors have turned their attention to data integration, few have addressed all the challenges, and many are creating solutions that are more complex than the original challenges. At the same time, many data consumers lack the skills necessary to take advantage of such solutions, which means involving more people in each integration project—as well as more time and expense.

## The Avaki solution

Avaki software implements a federated approach to data provisioning and integration that leaves distributed data in place and provisions it to users and applications across the organization. Many companies today prefer federated solutions, which leave data in place, to “big bang” solutions that require moving data into a central repository. Federated solutions minimize costs associated with disrupting data, users, applications, and administrators.

When you install Avaki software you create a unified, low-overhead system for provisioning distributed data across departments, locations, and companies. This system is called a data services layer or data grid. The data service layer’s scope can be small or large. It can serve one department or an entire extended enterprise.

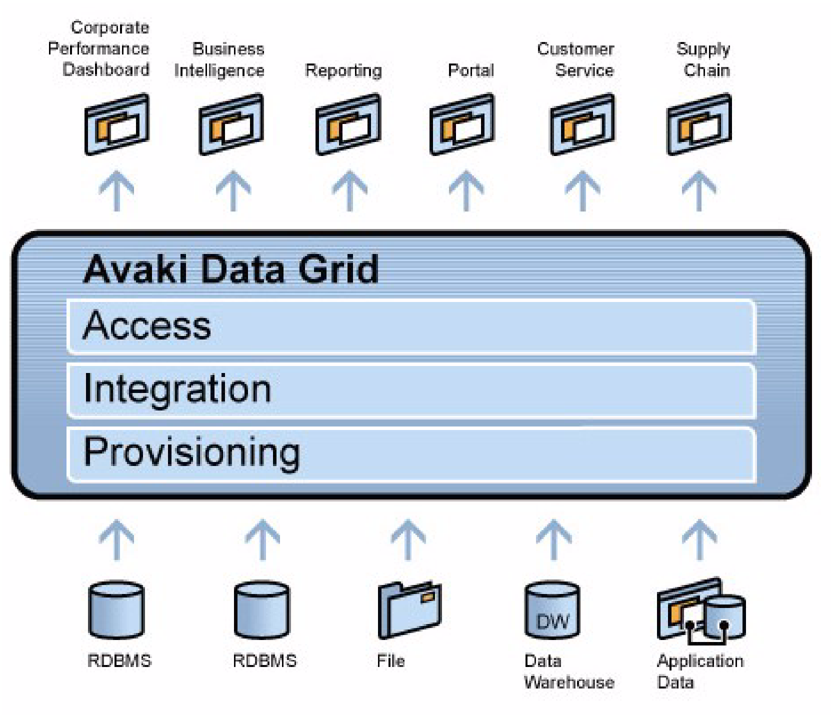


**FIGURE 1.** Avaki retrieves data from multiple sources of different types, tailors it in ways users and developers need, and makes it available securely across the organization. Users and applications access data through standard interfaces and do not need to know where data is physically stored.

A data grid provides:

- One data services layer where users and applications can access the data they need—Avaki supports both read and write access to data
- One unified catalog that enables access to multiple types of data: relational data, XML documents, file data, and application data
- One unified access control mechanism that operates across networks, locations, departments—even companies

How does it do this? First, you install a set of server components on your existing network, creating a *data grid* that you can think of as a large catalog you might use to "shop" for data. At the beginning, the catalog is empty. Then, one by one, individual data owners "publish" their data for others to use, creating entries in the catalog. At the same time, they establish access rights for each catalog entry, specifying who can read the data, who can update the data, and so on.

In creating entries in the data catalog, data owners do not create replicas of their data. Instead, they create a link from the data catalog entry to data that exists somewhere—in a production database, an operational data store, a data warehouse, or a file server—wherever data is currently stored and managed.

The data catalog's entries are arranged in a hierarchy much like any other directory structure, with one important difference—the data catalog is location independent. Users and developers do not need to know where data is physically stored in order to find and use it. They have *one place to go* to retrieve all data available to them.

But what about impact on the data source as new users and applications add to its load? By making explicit *which* queries are allowed to run against a data store, and by controlling cache coherence windows or prescheduling queries to run at specific times or with a certain frequency, data owners can control the load on operational systems. Avaki has rich caching and scheduling capabilities that make this process easy to administer and transparent to the consuming users and applications.

## Data access

How can a data owner in one location grant access to a user somewhere else? Through a unified access control mechanism that operates across networks, departments, and locations. Once we create our catalog, we want to give many different users and application developers rights to shop there and find all the data they need.

Avaki integrates with your existing directory services. If users and groups are under different administrative domains within the organization, you can integrate each local directory service that authenticates those users. A single Avaki domain can integrate with multiple, heterogeneous directory services including LDAP-based directory services (Microsoft Active Directory®, Sun ONE Directory Service®) and Network Information Service (NIS) directories. Because Avaki can integrate with multiple, heterogeneous directory services, you can associate many different users and groups with an Avaki domain without disrupting local authentication schemes. In this way, you create a virtual pool of users and groups to whom data owners can grant access to data. And you do so without having to give remote users accounts on each local system.

For a user to have access to a specific data item, the data owner must establish the appropriate access rights for that item. The user can then browse the catalog to find the data available to him or her, and access the data as if it were local.

Consider some important implications of this unified approach:

- Users no longer need to have direct access to multiple networks and databases to obtain data. Instead, they have a single sign-on for all data, and one place to go to get that data.
- Users and applications do not have to know or specify where data is physically located in order to access that data. They only have to know how it is named in the catalog.
- Users can search across the entire catalog to find the data they need, regardless of how many data owners or locations are involved in provisioning data, or how many data sources are involved.
- Applications have *one place* to access any data they need, and one standard set of interfaces for doing so, regardless of the specifics of the data source.
- There is no disruption to local security practices. When users access Avaki, they will continue to be authenticated by their local networks, and their local administrators will still have responsibility for them.
- Because existing data has not moved or changed in any way, existing applications that access local data do not need to change.
- Because Avaki links to original data rather than copying it, data is always up to date, never out of sync—and storage costs are reduced.

data catalogs provide a logical representation of a physical data source or data service.

## Data catalog entries

What, exactly, do data catalog entries represent? Each catalog entry is a logical representation of a physical data source or data service. There are several types of entries:

- **Database operations** provide access to some specific data from a relational database.
- **Data services** give your applications broader access to data from relational databases, files, and HTTP-based data sources such as a CGI scripts, servlets, or web services—and also perform transformations.
- **SQL views**, each representing a data service, a database operation, or a table in a relational database—and like tables, SQL views respond to SQL queries.
- **View generators** can convert provisioned or transformed data into a formatted output file in the data catalog.
- **Files** that exist somewhere on a file server—these can be data files, XML documents, XSLT style sheets used for data processing, and so on.

**Database operations.** A database operation is the vehicle through which users and applications have access to data in relational databases. Each database operation accesses one relational database. The data owner (typically a database administrator who has responsibility for that database) creates the database operation as an entry in the catalog, giving it a name and a definition.

The definition can be any SQL statement the database accepts (INSERT, SELECT, UPDATE, DELETE, CALL). When you define a database operation that contains a SQL statement and grant access to that database operation, you are granting rights to run that statement in the database.

Database operations can accept and return parameters as well as SQL result sets. For example, you could set up a statement that returned order details given a specific order number, and then supply the order number at runtime.

Avaki supports distributed transactions using the two-phase commit protocol. You can configure a set of database operations to be executed as a group (using a data service).

A database operation whose source database is the Avaki virtual database is called a virtual database operation. Use virtual database operations if you want to encapsulate and reuse SQL SELECT queries against Avaki SQL views.

The results of a database operation can be cached either on-demand or in a scheduled manner—in one or more caching services—thereby reducing load on back-end databases, decreasing latency, and controlling use of network bandwidth. Cache coherence is controlled with configurable coherence windows.

**Data services.** Data services are powerful integration tools. Like a database operation, a data service provides access to data for end users and applications. But it is not limited to a single database, or even to database sources. A data service can combine data from multiple sources, including database operations (for database data), files, HTTP sources, and other data services. Data services can also transform and filter data using relational operators like select, aggregate, join, and projection. Further, you can set up a data service that executes several database operations as a single *distributed transaction*.

Once set up, a data service is reusable across applications—an important benefit to development productivity.

There are several ways to set up a data service. The easiest is to use Avaki Studio to create a *view model*, which you can deploy as one or more data services. You can also create a data service by writing a *data service plug-in* in Java, JavaScript, or XSLT.

Like the results of database operations, the results of data services can be cached throughout the Avaki domain for improved performance.

**SQL views.** A SQL view is a virtual table; it's a data catalog entry that represents a table in a relational database, an Avaki database operation, or an Avaki data service. SQL views can be provisioned directly or generated from database operations or data services. SQL views are treated like relational tables by the Avaki query engine, and you can execute SQL queries against them. All the SQL views in an Avaki domain make up the domain's *virtual database*.

**View generators.** In many situations, the ultimate consumer of data that has been provisioned or processed is an application or end user who needs the data accessible as a flat file in CSV, HTML, or XML format. That function is fulfilled by Avaki views.

**Files.** A data owner with a set of files can "publish" the files by sharing them with the Avaki domain, naming them in the data catalog structure, and specifying access rights. Typically the data owner publishes an entire directory. Applications and users who need the file can then access it through the data catalog without having to know where it is located.

Files that can be shared include direct-attached storage, network-attached storage, or SAN storage. Avaki can handle very large files.

### Application data

Application data can exist in a variety of forms. For many applications, data can be made available to the data catalog without programming. Some applications that use relational data provide an interface with database views that can be accessed using SQL. Data from these sources can be made available via database operations as described earlier. Data from many applications with HTTP interfaces can be made available by creating a data service that references the URL and granting access to it. Data from applications that have neither a database interface nor a URL-based interface, and that make data available only through a proprietary API, can be provisioned into Avaki via data services.

### Integration made easy

Avaki provides a powerful graphical integration tool: Avaki Studio. Using an intuitive point-and-click interface, data consumers can identify data sources, build *view models*—data flows that specify how to combine and transform data from particular sources—and deploy view models as data services. No programming is required.

Avaki Studio's predefined operators let you perform relational operations like select, join, aggregate, group by, and order by, as well as advanced operations including iterate, which can perform a computation for each input in a specified range; update, which can insert data into a relational database; and generate, which can create new result sets. You can define custom operators to provide any further functionality you need.

In addition to tools for building data services, Avaki Studio provides tools for working with *metadata models*. A metadata model represents the schemas (columns and data types) of one or more database tables. You can use a metadata model to enforce a schema—a view model derived from a particular metadata model will share the metadata model's schema, and Studio will alert you if the schemas diverge. You can be sure that any data service deployed from such a view model will conform to the original schema.

Metadata models also let you create Avaki objects like data services that match database tables not only in schema but in name, so you can easily access the Avaki objects using the same applications you now use to access database tables.

### Protecting production databases

Because users and applications have access to the data from the original source, you might be concerned about how this will affect production systems. After all, database administrators need to ensure that production applications they support will perform well. Avaki is specifically designed to help database administrators insulate their production databases from risk in several different ways:

- **By providing access to data through stored procedures**. Stored procedures are totally under the control of the DBA and have been optimized for the database.
- **By limiting access to all but specific, predefined queries.** Database administrators provide the queries used by users and application developers, and can define and test queries that will perform efficiently before making them available.
- **By caching results.** Access to relational data does not have to be dynamic as long as users and developers can retrieve reasonably fresh data. Avaki enables database administrators to cache results—either on-demand or on a pre-determined schedule—so users never need direct access to the database. Results can be cached as frequently as required to meet the needs of data consumers. Avaki caching has no impact on applications; no special coding is involved in using cached data.
- **By enabling access through data services and generated views**. When users need data in other forms, DBAs can provide generated views or data services, then grant access to those objects. The views or data services can be updated on a schedule, or updated when the database operation is updated.
- **By using two-phase commit for distributed transactions.** You can build a data service that uses the two-phase commit protocol to execute a group of database operations as a single transaction. This ensures that related operations that perform work on multiple independent databases will leave all the databases in a consistent state.

## Ensuring data security

Avaki helps protect intellectual property and sensitive data by ensuring that only users who are authorized to access specific data can access it. With Avaki, you do not have to give a remote user an account on your network in order for that user to have access to specific data on that network. In addition, you do not need to tell users where data is physically located in order to give them access to it. Here are some of the ways Avaki protects intellectual property and confidential data:

- By providing a consistent, orderly, managed way to access data, which is a great improvement over ad hoc solutions
- By providing fine-grained controls that implement access policies in a consistent manner and are modified easily in response to policy changes
- By integrating with directory services for authentication of users
- By integrating with firewalls via HTTPS for secure operation across locations

- By encrypting all Avaki communications via SSL
- By giving data owners an SSL-based encryption option for data transmissions
- By providing audit logging capabilities that can be configured to record and/or issue alerts for events like file accesses, file writes, or data service executions.

### Data representation

Avaki software has several flexible capabilities, such as data services and view generators, that let you manipulate data in any format that's appropriate for your application. However, Avaki uses SQL rowsets and XML as the primary means for representing data. A rowset is a self-describing sequence of rows, or tuples. Each row consists of several named and typed columns.

If you are accessing Avaki via ODBC, you will work with the corresponding ODBC abstractions.

Avaki also includes support for representing and manipulating data using XML. In data services and view generators, for example, you can use XSLT to perform data integration and transformation operations.

A unique capability of Avaki is the ability to transform rowsets into XML on demand and to use either rowsets or XML as batch input to database operations. This conversion is on the fly; the XML is produced from the rowset as needed. Thus, a large rowset rendered in XML does not need to be represented in memory all at once, either as a rowset or as XML.

### Reusing provisioning work

You can see that with Avaki only a small amount of work is needed to make data available to users and applications. In addition, all the work you do can be reused. If data is made available for one purpose, and the same data is needed by another group or application, the data owner need only change the access rights on the catalog entry.

# *Data access*

Applications, application developers, and end users all need access to accurate, consistent, and current data in order to do their jobs. Applications need data to fuel business processes and provide information to decision-makers. Application developers need to integrate data into their applications. Knowledge workers need access to data to make a thousand small and large decisions every day.

Developers and integrators spend countless hours today finding or managing data, and countless hours writing code to integrate, aggregate, and transform data from multiple sources around the organization. This time could be spent on more productive activities if only there were a fast, easy way to get access to data. For developers this means simple, industry-standard APIs that provide data in the form needed by applications.

## The Avaki solution

Avaki software helps you simplify data access by providing one unified data layer with a single data catalog. With this unified approach:

- Users and developers have one place to go to find what they need, and do not need to know where data is physically located. Users and developers can search across the entire data catalog to find the data they need.
- Applications have one way to reference data—by catalog entry. They do not need to hard code data locations. References to catalog entries are resolved dynamically at run time.
- Data is available using a number of protocols and interfaces so users and applications can use the means that is most appropriate for their end.
- Applications have one standard set of interfaces for accessing data (one JDBC driver, for example) regardless of the specifics of the data source.

### Real-time access

Data owners can grant access to original data while using caching and other mechanisms to safeguard production systems. As a result, data can be made available to users and applications much more efficiently. Data owners can choose the frequency of cache updates based on the importance of fresh data to the data consumer’s business goal.

### Data access for users

Users who will only be accessing data and do not need other Avaki features can have completely transparent access to Avaki data through standard file system protocols. They will see only their standard file system and do not even need to know they are accessing data through Avaki. Users can also take advantage of Avaki's web user interface for searching and browsing. Though many users will be taking advantage of Avaki through an application, it is possible for users to access data directly. For example, a data owner could provide some data from a relational database in a comma-separated value format that a user could open using Microsoft Excel.

### Data access for applications

Applications can access data in a variety of ways depending on their specific need:

- **ODBC and JDBC**. Database applications accessing relational data will typically use an ODBC or JDBC interface, making a call to a database operation or a data service just as they would make a call to a stored procedure in the database, or querying a SQL view as if it were a table in the database.
- **File Read/Write.** Applications that expect data in flat file form or XML format can simply read the file by referencing the appropriate entry in the data catalog. Like end users, applications can take advantage of standard file system protocols (NFS for Unix, or CIFS/SMB for Microsoft Windows) to read the files transparently.
- **Web Services/SOAP.** Applications can make SOAP requests to perform Avaki actions (for example, requests for data), accompanied by the appropriate user authentication.

### Searching and metadata

One benefit of a unified data catalog is that users can search the entire set of objects or any meaningful subset. Each data object has attributes that store information such as the time the object was created, the name of the owner, the last time the object was modified, and so on. These *system attributes* are automatically defined for all objects. In addition, data owners or users can create custom attributes for any data item for which they have write permission, and specify values for those attributes. Because users can search the data catalog based on attribute values, you may want to create attributes that you know users will eventually want to search on. For example, you could define a *Project* attribute and assign values for individual projects that would allow users to search for data associated with a specific project. When users search the data catalog, they see only data objects for which they have at least read access.

## Caching in a nutshell

Avaki uses caching to accomplish a few different goals:

- Insulate production data sources from haphazard access
- Maintain good performance for users and applications
- Refresh data in a granular way based on business need
- Ensure maximum data availability

Caching makes remote data access practical by limiting the number of times a data request requires immediate communication with the original data source. Avaki provides a variety of different caching options and features to meet diverse performance requirements. You can specify a different caching strategy for each data item, and caching options can be used separately or in combination to accomplish your goals.

- **Local caching** enables caching of frequently requested results near the data source to reduce load on the back-end data source.
- **Remote caching** caches data close to the users or applications that will use it. This cuts down on network congestion and dramatically speeds up application performance, because remote data calls are eliminated. Caches can be pre-populated and updated during off hours when network load is low, and cache configurations can be established that ensure high availability when a network is congested or unavailable for some reason.

Cache update frequency can be specified for a given data item. Database administrators can schedule how often database operations should be re-run and cached, so as to protect production databases from unexpected load. Cached data can also “expire” after a set time period, forcing a refresh of the data on the next request.

Avaki’s caching mechanism is efficient and simple to deploy:

- It provides one solution for multiple data types, including multiple heterogeneous database management systems, files, XML documents, and application data
- No special hardware or network infrastructure is needed
- No additional database licenses are needed
- No application changes are needed to take advantage of the cache

An automated caching approach means fewer data stores to manage. Once you have set up specified caching options for a given data item, there is very little work to do. Caches are updated automatically in accordance with your specifications, and the process is completely transparent to users and applications.

### Auditing and compliance

Avaki's superior audit logging capabilities can help ensure the security of your customer information and other data while at the same time enabling you to comply with legal requirements for internal/external control systems. Administrators can use log4j, an open source logging package, to configure logging of any events that access, execute, or modify items that have been provisioned into or created in the Avaki data catalog—file accesses, or the execution or creation of database operations or data services, for example. Events can be logged to files, external databases, or both. You can even issue alerts (such as e-mails) to notify administrators when particularly sensitive items are accessed. Events logged to databases can be captured with business intelligence (BI) reporting tools (such as Crystal Reports and BusinessObjects) and then used to generate documentation that satisfies auditing requirements.

## *Create a scalable solution architecture*

Creating a solution that truly operates across locations and departments—even across companies—has its challenges, and some of them may not be obvious. The way an information integration solution is designed can seriously affect its ability to serve an extended enterprise as well as an individual department.

### The Avaki solution

Avaki software provides a flexible and scalable information integration solution based on a distributed, pluggable architecture. Sybase's expertise and experience with grid technology have led to an architecture design that addresses the key challenges thoroughly. Grid technology is all about distributed architectures and sharing of resources, giving Avaki a unique advantage and strength in data provisioning and access across departments, locations, and companies.

With Avaki, each administrator manages an *Avaki domain*. While the administrator has total control over his or her own domain, there is no need for a central administrator who controls all domains. Instead, administrators can create interfaces between domains that enable data owners in one domain to grant access to users or groups in the other. Sharing can be very broad or very limited, depending on business need. This approach enables sharing of data over an arbitrarily wide area, but maintains local administrative control over data sources, user management, and administration of the domain infrastructure.

### Adding capacity

Avaki administrators can add capacity as needed by plugging in more server components. They can address new requirements by adding users, data sources, and integrations to the grid. Software installation is easy. No specialized hardware is required, and in many cases, server components can be installed on existing infrastructure. Optional failover configurations make Avaki a resilient solution that provides for maximum data availability.

### Ease of administration

Once installed, Avaki is very easy to administer. A simple web interface lets administrators perform common administration tasks, including:

- Setup tasks such as proxy server configuration, directory service integration, and failover configuration
- Data administration tasks such as making data available and setting access rights
- Routine tasks such as monitoring, logging, and backups

While Avaki domains are surprisingly easy to manage, the biggest administrative benefit comes from reduced infrastructure complexity and from all the tasks that no longer need to be performed.

## *Supported platforms*

Avaki software runs on the hardware/operating system platforms listed below. For information on supported versions, as well as memory, disk space, and other system requirements, see the *Sybase Avaki EII Administration Guide*.

For Avaki servers (all types) and command clients:

- Intel/Red Hat Enterprise Linux ES
- Intel/SuSE Linux
- Intel/Windows 2003 Server, Windows XP Professional
- IBM AIX
- SPARC/Solaris

For Avaki Studio: Windows 2003 Server, Windows XP Professional

For NFS clients: NFS version 2 or version 3 client software

For CIFS clients: Windows 2003 Server, Windows XP Server or Professional

Avaki software is not required for NFS or CIFS clients.

## *A typical Avaki deployment*

An Avaki domain consists of one or more servers that together implement the data catalog and provide Avaki's data integration framework and its provisioning and access services.

Avaki domains can be accessed by a number of different clients. In some cases, clients require no Sybase software installed on their machines. This category includes transparent file access clients that access files in the data catalog via NFS or CIFS and web service clients that access Avaki via SOAP calls. Clients that require some Sybase software installed include ODBC/JDBC clients and machines on which people use the Avaki command-line interface (CLI) client.

### Getting started

Installing and starting a grid domain controller, the first server in an Avaki domain, takes only a few minutes. Your Avaki deployment architect will help you plan and deploy additional servers if you need them. (See "Avaki servers," below, for more information.) Setting up user accounts is easy—you can create them in the Avaki domain or import them from an NIS or LDAP-based directory service.

**Provisioning:** When you're ready to bring data resources into your Avaki domain, Avaki's provisioning tools in the web UI and in Avaki Studio walk you through the process of creating database operations (SQL queries or stored procedure calls), sharing file data, and provisioning web services for easy access.

**Integration:** Avaki Studio provides a set of drag-and-drop graphical tools you can use to design data flows and set up Avaki data services for integration. Even complex flows with many operations are simple to create. You can pull in data from multiple database, HTTP, and file sources, and combine and transform the data using a set of predefined operators such as aggregate, projection, group by, multiplexer, and iterator—or define your own operators.

You can also use Avaki Studio to work with *metadata models*. You can use metadata models to ensure that new data services you create conform to a particular schema. Metadata models also let you create Avaki objects like data services that match database tables not only in schema but in name, so you can easily access the Avaki objects using the same applications you now use to access database tables.

You can even perform some provisioning tasks from within Studio, such as defining database connectors and database operations.

## Avaki servers

A machine that participates in an Avaki domain can host one or more Avaki servers of several types. Start with a basic Avaki domain and add servers with advanced capabilities or to increase capacity, as required.

Figure 2 shows a typical deployment that uses all server types.

**Basic Avaki domain.** A basic Avaki domain might contain just one or two grid servers, one serving as the grid domain controller (GDC). Additional grid servers can be added to accommodate more data, data services, or additional sites.

- Grid server: hosts the data catalog, provides authorization services for clients requesting data access, serves files shared from the local file system, and runs data services, database operations, and queries.
- Grid domain controller (GDC): the grid server on which an Avaki domain is initially started. The grid domain controller has all the functionality of a grid server. An Avaki domain must have at least one grid domain controller.

  In an Avaki domain that is configured for failover, there are two GDCs: a primary and a secondary. The secondary GDC is a hot standby that handles requests when the primary GDC is unreachable.

**NFS or Windows file access.**

- Data grid access server (DGAS): provides high-performance caching and makes specified Avaki directories available to NFS and CIFS (Windows) clients in a secure fashion.

**Extended file sharing.**

- Share server: makes selected data stored in local file systems visible in the data catalog. Share servers are responsible for file I/O. Each grid server can be associated with several share servers.

**Interconnecting domains.**

- Proxy server: allows Avaki domains on opposite sides of a firewall to communicate securely with one another so that users of each domain can access data in the other.

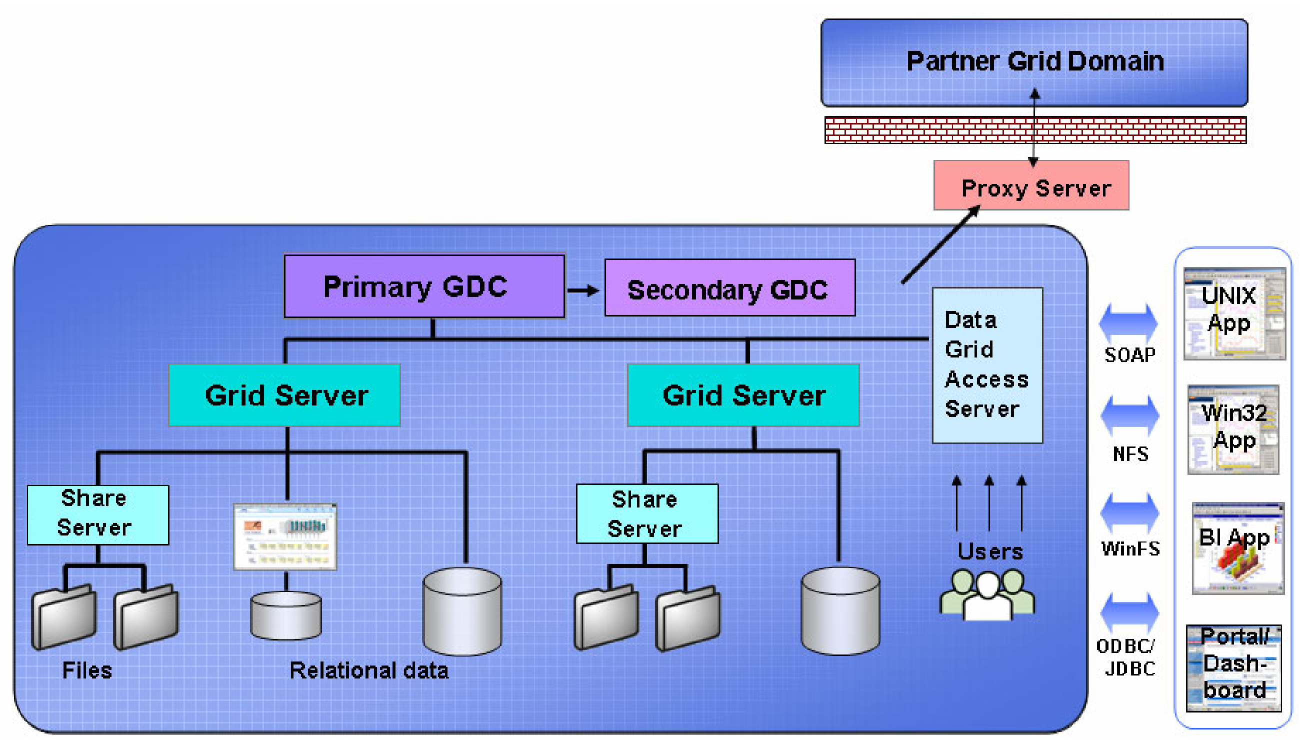


**FIGURE 2.** An Avaki domain with primary and secondary GDCs, grid servers, share servers, a proxy server and a DGAS deployed. Users (lower right) and applications (far right) can access relational data and web services via Avaki services configured on the grid servers. They can access files via the DGAS using NFS or CIFS (Windows) clients.

## Clients

An Avaki client can be one of the following:

- Command-line client: lets administrators and advanced users issue Avaki commands. A command client provides no storage resources to the domain.
- NFS client: mounts the Avaki data catalog (or a subset of it) as a directory by connecting to an Avaki data grid access server.
- CIFS client: accesses Avaki directories that have been shared out by connecting to an Avaki data grid access server.

- ODBC/JDBC client: on a machine where applications need access to database data, you can configure a Sybase JDBC driver and program the applications to retrieve the information via JDBC.
- Avaki Studio: lets data architects build and test data flows and deploy them as Avaki data services.

Chapter 2

# Database tools

This chapter introduces the Avaki tools you'll use to work with dynamic data. The tools fall into two areas:

- Provisioning: Getting data into the data catalog (from files, databases, and HTTP sources such as web services and CGI scripts) and setting appropriate access controls. The section on provisioning, which focuses on extracting data from databases, begins on page 22.
- Integration: Processing, combining, and transforming data into the formats required by users, applications, and databases. This might involve data provisioned from multiple data sources as well as data derived from other data integration steps. The section on integration begins on page 23.

Once the source data and the various integration services have been provisioned and defined in the data catalog, end users and applications can access or invoke these data sources and services using several means: Transparent file system access via NFS or CIFS, web services (SOAP, the Simple Object Access Protocol), Open Database Connectivity (ODBC), or Java Database Connectivity (JDBC). You can also access data sources through Avaki Studio and use them to create data services. When considering the various means for provisioning data and implementing integration operations, you will want to consider which end users or applications will need to access it. One of the many strengths of Avaki is that you can make the same data accessible via a number of interfaces or protocols, thereby serving multiple constituents without having to perform any extra work.

# Provisioning

Avaki database connectors, database operations, and SQL views make database data accessible to users and applications. These tools are discussed in the subsection that follows.

## Provisioning database-resident information: Database connectors, SQL views, and database operations

To make relational data accessible to Avaki, you must create an entry in the data catalog that represents a connection to a database. This kind of catalog entry is called a *database connector*. A database connector encapsulates information such as the JDBC driver, connect string, and username and password that Avaki will use to connect to the database, as well as other useful information such as how to contact the appropriate database administrator.

Once you establish a connection to a database, you can make data from that database available to Avaki in the following ways:

- **Provision SQL views.** When you provision a *SQL view*, you make a table in a provisioned database known to Avaki. Avaki extracts the metadata for the table and uses it to create an object called a SQL view in the Avaki data catalog. Provisioned SQL views (or tables) are treated like relational tables by the Avaki virtual database's query engine. When you create connectors to one or more databases and provision SQL views from each of those databases to Avaki, you create a single *virtual database*: You can execute SQL queries on these tables as if they were tables in a single database. Avaki's query engine will analyze these queries, push as much of the work as possible down to the actual database, and perform the remaining operations (such as joins across tables from different databases) itself.
- **Create database operations.** A *database operation* is the vehicle through which users and applications have access to data in relational databases. Each database operation is a data catalog entry that accesses one relational database and encapsulates a single SQL statement, such as CALL (a stored procedure invocation), SELECT, INSERT, UPDATE, or DELETE. Database operations can take parameters. When you connect to Avaki through ODBC or JDBC, database operations appear and can be invoked as stored procedures (using SQL CALL syntax). They can also be invoked via SOAP.

  The output of a database operation is one or more SQL result sets that conform to the Java rowset abstraction.

Avaki will generate a schema that represents the output result set of the operation. You can view this schema either from the web UI or from Avaki Studio, or you can access it via JDBC. You can also generate SQL views from database operations that do not update the underlying database.

Use database operations to encapsulate optimized, reusable queries against an underlying database. Database operations, in addition to being invocable directly, can be used as input sources for data services that you model and deploy from Avaki Studio.

Avaki database connectors, SQL views, and database operations are typically set up by database administrators. You must be a member of the DatabaseAdministrators group to set them up.

## Integration

Avaki appears to external applications and users as a virtual database when accessed via JDBC or ODBC. As noted previously, the simplest data integration tasks can be accomplished simply by using SQL SELECT syntax to run queries against SQL views that have been provisioned in Avaki.

However, in many cases, access to Avaki will not be entirely ad hoc against "raw" provisioned SQL views. For example, you might want to expose a logical model that involves creating a combination of procedures and virtual tables that encapsulate operations against raw provisioned SQL views and database operations. The primary mechanisms for building up this logical layer of abstraction in Avaki are as follows:

- You can use Avaki Studio to define *view models*, and then deploy those view models as *data services* within Avaki. Like database operations, data services are accessible as stored procedures via ODBC and JDBC. They are also accessible via SOAP. In addition, you can generate SQL views from data services that return rowsets, thus exposing these logical abstractions as tables as well as stored procedures via SQL.
- You can create *virtual database operations*. A virtual database operation is just like a regular database operation, except that its source database is Avaki itself. Use virtual database operations if you want to encapsulate and reuse SQL SELECT queries against SQL views (provisioned or generated from database operations or data services) within Avaki.

To cover more advanced situations where you need the full power of Java or you need to perform XML processing within Avaki, you can also create data services that are implemented in Java, JavaScript, or XSLT directly without using Avaki Studio.

Finally, if you have data consumers or applications that require flat file output, you can route the output of database operations or data services to flat files using Avaki view generators.

## The Avaki data catalog viewed through ODBC/JDBC

To refer to an Avaki SQL view (table), database operation, or data service via ODBC or JDBC, you must know its name. These objects have three-part, qualified names in the following format:

```
<catalog>.<schema>.<object>
```

<catalog> is the name of the Avaki domain in which the object is defined. The <schema> portion of the name can be one of the following:

- A database connector name. For example, if you establish a database connector called “CustomerDb,” then all SQL views and database operations defined on that connector will have CustomerDb as the schema portion of their name.
- The literal “DATASERVICE.” This is the schema in which all data services and SQL views generated from them are deployed.
- The literal “VirtualDb.” This is the schema in which virtual database operations and SQL views generated from them are deployed.

<object> is the name of the SQL view, database operation, or data service that you want to access.

For more on qualified names of Avaki objects, see “Qualified names: shorthand for paths” on page 30.

## Processing distributed data via XSLT, JavaScript, and Java: Advanced data services

A data service can process data from one or more sources of the following types:

- files (which can be *generated views*);
- external web services;
- other data services;

- Avaki *database operations* or *virtual database operations*.

Data services can be invoked via JDBC or SOAP calls, or they can be invoked as input to other data services and view generators. A data service accepts runtime parameters, which can be routed to the input sources or to the data service plug-in. The output can be in rowset, XML, or raw byte stream format; output may also include parameters.

Data services perform their processing in a *data service plug-in*. A data service plug-in can be implemented as an XSLT style sheet, as JavaScript, or as Java classes (packaged in a JAR file). You can use Avaki Studio to generate and deploy data services whose plug-ins are written in JavaScript; see *Data Integration with Sybase Avaki Studio* for details. (Note that when you use Studio to create a data service, you may never see the plug-in—Studio creates it for you.)

If you want to write your own data service using XSLT or Java, you'll use a different set of tools. The style sheet or JAR file must be shared into the data catalog. See the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide* for instructions on creating data service plug-ins in XSLT or Java.

An important application of data services is to support *distributed transactions*. When you need several database operations to be treated as a single transaction—that is, all must succeed, or if any fails, all must be rolled back—you can write a Java plug-in that uses the two-phase commit protocol, then set up a data service that calls all the database operations. See the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide* for instructions.

Data services are configured by members of the DataProviders group.

## Generating data snapshots as files: Avaki view generators

In many situations, the ultimate consumer of data that has been provisioned or processed in the grid is an application or end user that needs the data accessible as a flat file, which might be in comma-separated values (CSV), HTML, XML, or another format.

Avaki views meet this need. They consist of two parts: the *view generator* and the *generated view*, which is the file that results from view processing. When you configure a view generator, you name an input source and specify the parameters that the input source requires, as well as where in the data catalog you would like the generated output to appear. Finally, you configure the kind of transformation that you'd like performed. By default, Avaki can simply pass through the results of the input and generate a CSV file or an HTML page. If you need customized processing, you can also specify an XSLT style sheet that will transform the input if it is XML (or rowset data that Avaki will convert to XML on the fly).

Views can take input from files (including other views), data services, and database operations.

When generated views are unused for a certain period, they are discarded.

You can configure views to issue *update notifications* whenever they are regenerated. Update notifications allow the regeneration of one view to trigger the update of another.

Views are configured by members of the DataProviders group.

Chapter 3

# Browsing the data catalog

This chapter describes the features, contents, and purpose of the Avaki data catalog.

In this chapter:

## The data catalog: What is it and what's in it?

The Avaki data catalog is the shared, distributed, hierarchical directory where data owners and administrators can register their data sources, files, queries and data integration services and set access control and other policies. Application developers and end users who need access to data use the data catalog to browse or search for the data sources or services they need, access the data directly, or invoke data integration services, subject to access control policies. The data catalog is distributed and can scale across an enterprise. By interconnecting Avaki domains, administrators can create data catalogs that span multiple enterprises while maintaining strict access control policies.

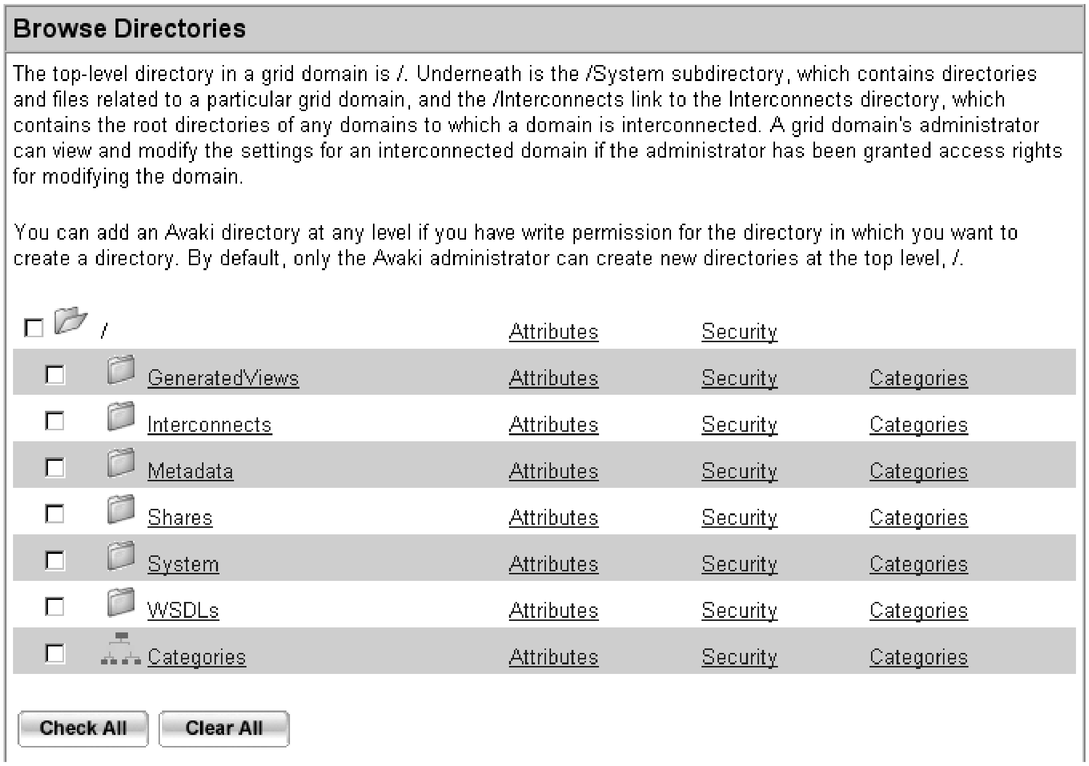
You can view and access the contents of the data catalog in various ways: through Avaki Studio's data catalog view pane, through the Avaki web user interface (web UI), through the Avaki command line interface (CLI), or via web services/SOAP.

Every object in an Avaki domain has an entry in the data catalog. This includes obvious things like data sources, files, grid directories (folders), database operations, service views, data services and view generators. It also includes servers, user accounts, authentication services, groups and search services. A core catalog structure is created with each Avaki domain; that structure can be extended and customized to suit your organization's needs.

Every object in the system therefore has a catalog name, or path. Avaki's APIs, commands and user interface also recognize so-called "qualified names" for certain types of objects—these are merely syntactic shorthand for catalog paths.

You can display the list of top-level Avaki directories (folders) in the web user interface by visiting the Browse Directories screen, shown below.

Home > Data catalog management



**Browse Directories**

The top-level directory in a grid domain is /. Underneath is the /System subdirectory, which contains directories and files related to a particular grid domain, and the /Interconnects link to the Interconnects directory, which contains the root directories of any domains to which a domain is interconnected. A grid domain's administrator can view and modify the settings for an interconnected domain if the administrator has been granted access rights for modifying the domain.

You can add an Avaki directory at any level if you have write permission for the directory in which you want to create a directory. By default, only the Avaki administrator can create new directories at the top level, /.

| | | | | |
|---|---|---|---|---|
| ☐ | / | Attributes | Security | |
| ☐ | GeneratedViews | Attributes | Security | Categories |
| ☐ | Interconnects | Attributes | Security | Categories |
| ☐ | Metadata | Attributes | Security | Categories |
| ☐ | Shares | Attributes | Security | Categories |
| ☐ | System | Attributes | Security | Categories |
| ☐ | WSDLs | Attributes | Security | Categories |
| ☐ | Categories | Attributes | Security | Categories |

## Icons

These icons identify different types of objects in the data catalog:

| | |
|---|---|
| ⚙ | A service (example: a database operation or an authentication service) or an Avaki server |
| 📁 | An Avaki directory |
| 🗄 | An Avaki share |
| 📄 | A file |
| ⊤ | A category or the link to the category browser |

## What is a data catalog name?

Anyone who has used a Windows or Unix file system is familiar with paths used to name files, such as:

```
C:\windows\winnt\pgm.exe
/home/local/jdoe/myfile.txt
```

The name of an object (file, directory or folder, database operation, etc.) in the Avaki data catalog is just like a Unix filename—a forward-slash-separated pathname:

```
/Shares/myShare/myfile.txt
/System/LocalDomain/Services/DatabaseServices/MyDb/Query1
```

Every object in a data catalog has a single unique pathname that is its primary name. However, you can use Avaki links to create secondary names for any object. Unlike hard links in a Unix file system, however, Avaki hard links are not reference counted—there is a primary link which, when unlinked, causes the underlying object to be deleted.

## Qualified names: shorthand for paths

There are situations where using full pathname syntax to call on a particular object in the data catalog is either not appropriate or inconvenient. For instance, when you invoke a database operation via a JDBC driver, JDBC dictates that you use a dot-separated three-part name syntax for the database operation. In other situations it's more convenient to use an abbreviated syntax to identify users. We refer to these syntaxes as "qualified names"—an alternative, usually shorter syntax for the full catalog pathname.

Here's an example of the qualified name for a database operation; it includes the names of the Avaki domain, the database operation's database connector, and the database operation itself:

MyDomain.MyDBconnector.MyDBop

Avaki expands such three-part names into data catalog paths of the form

/System/Domains/*MyDomain*/services/database services/*MyDBconn*/*MyDBop*

If the domain name is omitted from the qualified name, LocalDomain is assumed.

Avaki recognizes qualified names for the following objects:

- Data services
  There are two ways to specify a data service for the web services API, the CLI, ODBC, or JDBC. You can always use this syntax:

  &lt;domain name&gt;.dataservice.&lt;data service name&gt;

  In addition, a data service that has been mapped to a table in a metadata model in Avaki Studio can be accessed using the metadata model and table names:

  [&lt;domain name&gt;.]&lt;metadata model name&gt;.&lt;table name&gt;

  For example, suppose there is a data service called empDS in the Bedrock domain, and empDS has been mapped to a table called Employee in the HumanResources metadata model. You can access the empDS data service using either of these names:

  Bedrock.dataservice.empDS

  HumanResources.Employee

- Database operations
  When you access a database operation through the web services API, the CLI, ODBC, or JDBC, use the following syntax:

  &lt;domain name&gt;.&lt;DB connector name&gt;.&lt;DB operation name&gt;

  In addition, as with data services, a database operation that has been mapped to a table in a metadata model in Avaki Studio can be accessed using the metadata model and table names:

  [&lt;domain name&gt;.]&lt;metadata model name&gt;.&lt;table name&gt;

- Virtual database operations
  When you access a virtual database operation through the web services API, the CLI, ODBC, or JDBC, use the following syntax:

  &lt;domain name&gt;.virtualDB.&lt;virtual DB operation name&gt;

- SQL views (tables provisioned or generated in the data catalog)
  When you access a SQL view through the web services API, the CLI, ODBC, or JDBC, use one of the following syntaxes:

  – For a SQL view provisioned through a database connector:

    &lt;domain name&gt;.&lt;db connector name&gt;.&lt;SQL view name&gt;

   - For a SQL view generated from a database operation:

     <domain name>.<db connector name>.<SQL view name>

   - For a SQL view generated from a data service:

     <domain name>.dataservice.<SQL view name>

   - For a SQL view generated from a virtual database operation:

     <domain name>.virtualDB.<SQL view name>

- Tables in relational databases
  A database table that has been mapped to a table in an Avaki Studio metadata model can be accessed using the metadata model and table names:

  [<domain name>.]<metadata model name>.<table name>

- Avaki servers
  In the CLI, wherever you are asked to provide a server name, you can pass in just the name—it's not necessary to give the entire catalog path.

- Users
  Instead of the full path to a user account, you can provide a qualified name using this syntax:

  username@<authservice-name>.<authservice-type>.<domain name>

  where <authservice-name> is the name of the grid authentication service to which the user belongs and <authservice-type> is one of Ldap, Grid, Nis. If a domain name is not provided, LocalDomain is assumed. Qualified names for users expand to data catalog paths of this form:

  /System/<domain>/Services/AuthServices/<auth-service type>/<authservice-name>/Users/<user-name>

## Top-level Avaki directories

When you create an Avaki domain, the system sets up the core directory (folder) structure of the catalog. This section describes the core structure and its intended use. Note that members of the Administrators group can create additional top-level directories.

| **Avaki directory** | **Description** | |
|---|---|---|
| /GeneratedViews | The default location for files produced by Avaki view generators. This directory is provided for convenience; it's intended to serve as the root directory for generated views, but you can place generated views elsewhere if you want. | |
| /Interconnects | When you interconnect your Avaki domain with another, the other domain's root directory—its "/"—is linked into your domain's /Interconnects directory. (The other domain's /System/Domains/<domain name> directory is linked into the same place in your domain's catalog.) | |
| /Metadata | /Metadata contains generated XML schema documents (.xsd files) that show the XML representation of the output schema of your database operations. Do not modify these generated files or the corresponding directory structure. /Metadata contains these subdirectories: | |
| | /DataService | A generated directory containing a subdirectory for each data service in your domain. Data services that produce resultset output store their schema documents here. (Running a data service generates its schema; you can also use Avaki Studio's generate schema utility.) |
| | /Database | A generated directory tree that reflects the structure of the database connectors and database operations in your Avaki domain. The structure is:<br>/Metadata/Database/<database-connector>/<database-operation>/<database-operation>.xsd<br>That is, there is a subdirectory for each database operation, and within that subdirectory there is an XML schema document that represents the shape of the information returned by that database operation. |
| | /Relational | A generated directory structure where query engines create and access metadata for SQL views. There is a subdirectory for each SQL view in your Avaki domain. The metadata stored here is intended to be understood by query engines only. |
| | /Standard | A convenience directory into which you can link other metadata (or create Avaki shares containing such information). |

| Avaki directory | Description | |
|---|---|---|
| | /User | A convenience directory into which you can link other metadata that's of interest to particular users (or create Avaki shares containing such information). |
| | /VirtualDatabase | A generated directory structure that stores metadata for virtual database operations. The structure is:<br>Metadata/VirtualDatabase/<grid-server>/<virtual-DBOP>/<virtual-DBOP>.xsd<br>The XML schema document (.xsd file) for each virtual database operation represents the shape of the information returned by that virtual database operation. |
| | /VirtualSchema | A generated directory structure that holds deployed metadata models. |
| /Shares | The default location for Avaki shares in your data catalog. Each Avaki share you create here appears as a subdirectory. This directory is provided for convenience; you can place Avaki shares elsewhere. | |
| /System | Avaki software uses this portion of the data catalog to keep track of the servers and services that comprise an Avaki domain. The structure is controlled by the software—do not edit files or add, remove, or rename files or subdirectories here unless you're instructed to do so by a Sybase representative.<br>/System contains an authentication service and these subdirectories: | |
| | /Domains | /System/Domains contains one entry for the current or local domain (its name is the domain name) and one entry for every domain to which the local domain has been interconnected.<br>In each domain directory are subdirectories for servers, services and views in that domain. |
| | /Interconnects | Same as the top-level /Interconnects directory. |
| | /LocalDomain | Same as the subdirectory for the current grid domain in System/Domains. |

| **Avaki directory** | **Description** | |
|---|---|---|
| /WSDLs | Contains Web Services Description Language descriptions for the Avaki services that you can invoke via web service calls. These WSDL documents allow you to access Avaki database operations, data services, and data catalog functions via web services.<br><br>You can create new subdirectories under /WSDLs to link in other WSDL documents that you want to share within your organization.<br><br>/WSDLs contains these subdirectories: | |
| | /AvakiServices | Contains the "built-in" services offered by Avaki. Using the web services described by the WSDL documents in this directory, you can access any part of the data catalog, read and write files, set and view access control lists, and invoke database operations and data services. See the *Sybase Avaki EII API Guide* for more on Avaki's WSDLs and web services API. **Note**: Do not change, move, or delete this subdirectory or its contents. |
| | /DataServices | In subdirectories named for Avaki domains (the local domain plus any interconnected domains), DataServices contains WSDLs for data services. |
| | /DatabaseOperations | In subdirectories named for Avaki domains (the local domain plus any interconnected domains), DatabaseOperations contains WSDLs for database operations |
| /Categories | Categories let you classify and organize the contents of your data catalog. Clicking **Categories** takes you to the category browser. For more information, see "Data categories" on page 37. | |

## *Organizing your data catalog*

An appropriately organized data catalog will be a valuable asset to your organization. By taking advantage of the Avaki data catalog to organize and categorize your data assets, you will

- Be able to group related queries, data services, files, and other data assets without having to replicate the data physically.
- Enable users to find relevant information more easily.
- Increase the potential for re-use of provisioned data assets and data integration services.

Using the catalog's directories and categories as grouping and classification mechanisms, you can create arbitrary taxonomies of your provisioned and integrated data assets. A given item can appear in any number of directories and categories, and you can build taxonomies of arbitrary depth and breadth.

### Using links

Use links when you want an item in the data catalog to appear in more than one directory. Creating a link to an item in the data catalog does not create a copy or replica of any data. It simply creates a new catalog entry that points to the same object. There is no performance penalty for accessing an object via a newly created link. The data catalog supports two kinds of links:

- A *hard link* is an alternate name for an item in the data catalog. Changes to the object's other names have no effect on the hard link: for example, 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 the original name. Because they are more robust, hard links are used most often; see the *Sybase Avaki EII Administration Guide* for instructions on creating them.
- A *soft link* is a pointer to a particular location (name) in the data catalog. If the object at that location is moved, deleted, or renamed, the soft link leads nowhere. Soft links are used infrequently; see the *Sybase Avaki EII Command Reference* for instructions on creating them.

## Data categories

Use the categories feature to set up taxonomies for data items in the Avaki data catalog. Categories can be thought of as a parallel set of Avaki directories into which you can link objects in the data catalog. You can arrange the category hierarchy in a way reflects your organization's taxonomy for provisioned data assets—making database operations, service views, data services, generated views, files and so on easier to find and use.

For example, you might set up a taxonomy as follows:

Category: BusinessEntities

> Category: Customers
> Database operations, service views, data services, etc. relating to customers
>
> Category: Orders
> Database operations, service views, data services, etc. relating to orders. Note that a getOrderInfoGivenCustomerId service view might appear under Customers as well as Orders.
>
> Category: Products
> Items related to products
>
> Category: Shipments
> Items related to shipments

Category: Metrics

> Category: Order Fulfillment
>
> Category: Sales

Category: ReferenceInformation

> Category: HR
>
> Category: Product Briefs
>
> Category: Sales Presentations

Category: Reports

For instructions on setting up categories in Avaki Studio, see *Data Integration with Sybase Avaki Studio.* For instructions on setting up categories in the web UI, see the *Sybase Avaki EII Administration Guide*.

## Organizing shared files

When you create Avaki shares to provision files into the data catalog, Sybase recommends that you put these shared directories under /Shares. This is a convention, not a hard requirement. As with any directory in the data catalog, you can create any kind of structure that you want under /Shares in order to organize your shared file systems as you deem appropriate.

Having done so, you can then expose portions of these Avaki shares in categories, as described in the previous section, or by linking them into a directory structure under /Shares or elsewhere in the data catalog.

For example, as part of making all of your organization's data assets available via Avaki using Avaki software as a uniform data services layer, you might want to create Avaki shares that contain items like:

- Sales collateral
- Generated reports
- Commonly used spreadsheets
- Commonly used software tools
- FAQs
- HR-related forms

You can share the source directories into the data catalog under /Shares and then create categories for these file assets that correspond to how you want people to find and access this information. For instructions on setting up and managing Avaki shares, see the *Sybase Avaki EII Administration Guide*.

## How is the data catalog distributed among Avaki servers?

Every entry in the Avaki data catalog—whether it's a database operation, data service, Avaki directory, file, user, group, authentication service, or other object—has a "home server." The home server is the Avaki grid server on which the information that Avaki maintains about that object resides—its ACL, attributes, etc.

When you first create your Avaki domain, a number of core services and objects are created on the grid domain controller (GDC) and registered in the data catalog. When you join a new grid server into an Avaki domain, a number of core services and Avaki directories are created on that new server and the top level Avaki directories on that server are linked into the appropriate places on the GDC.

In some cases, such as when you create a database connector or a data service, Avaki gives you the option of choosing the home grid server for that object. In other cases, the choice is implicit—for example, when you create a database operation, its home server is the same as the home server of the database connector with which it is associated.

Thus, the data catalog as well as all the services and objects that it contains are distributed, not replicated, among the various servers that comprise your Avaki domain.

When you create an Avaki directory in the web UI, the directory is created on the same grid server as the directory that contains it. Thus, the taxonomies that you create via the UI will all reside on the GDC, since the root of the data catalog is on the GDC.

Chapter 4

# Authentication and access control in Avaki domains

This chapter discusses the mechanisms that protect an Avaki grid domain from unauthorized access. It covers the following topics:

- "Authentication," below
- "User accounts and groups" on page 43
- "Access control lists" on page 45

## Authentication

This section discusses authentication of login access to Avaki, file access via DGAS, and database access.

### Authentication of login access to Avaki domains

Before you can perform administrative tasks or provision or manage data resources, you must log in to an Avaki domain. The tasks you are allowed to perform depend on the permissions granted to your user account. ("Access control lists" on page 45 discusses ACLs and permissions.)

To log in, you must have a user account that's known to the Avaki domain. There are two types of accounts:

- Grid accounts
  Administrators create grid accounts directly in the Avaki domain. When a grid user tries to log in, password checking is performed by the Avaki domain.
- Imported accounts
  Administrators can import user accounts from an LDAP or NIS directory into the Avaki domain. When an imported user tries to log in, the Avaki domain passes the user name and password to the directory server, which performs the authentication.

Avaki uses three types of *authentication services*:

- Grid
  The Grid authentication service in each Avaki domain, which is called Default-AuthService, authenticates users whose accounts were created in that domain (grid accounts).
- Ldap
  LDAP authentication services authenticate users whose accounts are imported from LDAP directory services.
- Nis
  NIS authentication services authenticate users whose accounts are imported from NIS directory services.

**Note** When you enter authentication service types in Avaki commands or JDBC connection properties, capitalize the types as show here—Grid, Ldap, and Nis.

To import users from an external LDAP or NIS directory service, an administrator must first integrate the external directory service into the Avaki domain. During this process, you provide information to the Avaki domain about the directory server. This creates an authentication service, the Avaki representation of the directory service.

The steps for integrating external directory services, importing and creating users and groups, and related tasks can be found in the *Sybase Avaki EII Administration Guide*.

## Authentication of file access via DGAS

File access via data grid access servers is controlled by Avaki's authentication system, plus a mapping scheme that links Avaki user accounts to accounts in the system that hosts the files. For details, see the *Sybase Avaki EII Administration Guide*.

### Authentication of database access

When you configure a database connector, a database operation, or a Java program to access a database to read or update data, you either provide a user name and password for a database account, or specify that the database access should use the credentials of the user who initiates the execution. The user name and password are passed to the database, which performs its own authentication.

## *User accounts and groups*

An Avaki user can be a member of one or more groups. A group, which is simply a set of users, can be added to access control lists and granted or denied permissions just like a user. When you have several users to whom you want to grant (or deny) access to a particular resource or set of resources in your Avaki domain, you can save time by adding all the users to a group.

Avaki supports two types of groups:

- Grid groups
  Grid groups exist only in the Avaki domain. Every Avaki domain includes some default grid groups, which are described below. You can create additional grid groups as needed.

- Imported groups
  You bring imported groups into the Avaki domain from an external authentication service.

The *Sybase Avaki EII Administration Guide* includes procedures for managing both types of groups.

### Default grid groups

When you set up a new Avaki domain, it includes several grid groups by default. Members of these groups have special privileges to facilitate Avaki's role-based administration scheme. The default grid groups are as follows:

- Administrators
  Members of the Administrators group can perform all administrative tasks, including user administration (see UserAdministrators, below) and database administration (see DatabaseAdministrators, below). Only members of the Administrators group can add or remove users in the Administrators group, change the ownership of any Avaki object that belongs to a member of the Administrators group, config-

ure Avaki servers, and integrate and delete external authentication services. Members of the Administrators group can do everything. They are not subject to the permissions in Avaki access control lists; they can read, write, delete, and change the ownership and permissions of any object in their Avaki domain. Putting a member of the Administrators group on an ACL deny list has no effect. By default, the Administrators group has one member, Administrator, who can add other users as needed. It is not possible to remove the Administrator user from the Administrators group or to delete the Administrator user.

- UserAdministrators
  Members of UserAdministrators can create, import, and delete user accounts and groups, add users to and remove users from groups, and change passwords.
- DatabaseAdministrators
  Members of DatabaseAdministrators can create, modify, delete, test, and view information about database connectors and database operations.
- DataProviders
  Members of the DataProviders group can create Avaki shares and CIFS shares; set up data services; set up generated views based on files, database operations, and data services; create search services, and create categories.
- DomainUsers
  Every user in an Avaki domain is a member of that domain's DomainUsers group. Members of DomainUsers have no special privileges by default.
- MessagingUsers
  To support messaging between Avaki domains, each domain has a MessagingUsers group and a MessagingUser user account. To use cross-domain messaging, a domain in a two-way interconnect must add the other domain's MessagingUser to its own MessagingUsers group. This allows the first domain to receive update notifications about changes to generated views from the other domain. (For details on setting up cross-domain messaging between interconnected domains, see the *Sybase Avaki EII Administration Guide*.)
- everyone
  This group lets you, in a single action, grant or deny access to a file, directory, or other resource to everyone who has access to this Avaki domain. (Note that this is the only group that can be used to deny access.) In effect, adding "everyone" to an allow list turns off access control—it allows access even to users who have not logged in. If "everyone" is on an allow list, denying permission to an individual user has no effect. Conversely, if "everyone" is on a deny list, allowing individual users or other groups permissions on that object has no effect. Permissions set using

the everyone group take precedence over all other permissions except those of administrators.

You can create as many additional grid groups as you need.

You can use groups and user accounts—both imported groups and user accounts and those created in the data grid—in access control lists (ACLs), which are described in the section that follows.

## Access control lists

This section explains how access control works. It covers the following topics:

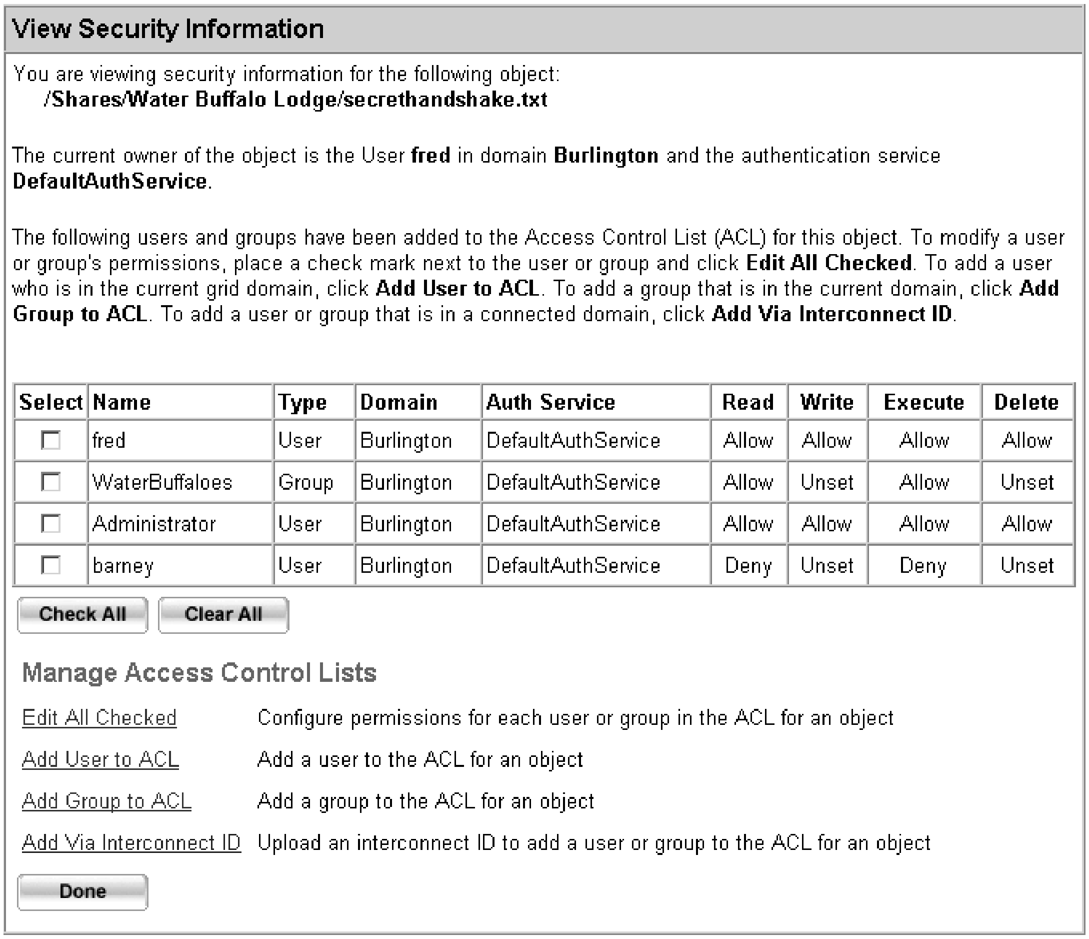
- "How Avaki permissions work," below
- "Ownership" on page 46
- "Permission settings" on page 47
- "Interpreting permissions" on page 48
- "Permissions in directories and categories" on page 49
- "Permissions on new files" on page 49
- "Permissions on cached objects: using groups" on page 51

### How Avaki permissions work

Access control lists (ACLs) determine which grid users are allowed to read and manipulate files, directories, categories, Avaki shares, database operations, and other objects in the data catalog.

Every Avaki object has an ACL that consists of a list of users and groups and the specific privileges of each user or group to perform actions on the object. The actions are read, write, execute, and delete. You can allow or deny permission for each action on each object by each user or group.

Here's a sample ACL; it's for a file called secrethandshake.txt.



View Security Information

You are viewing security information for the following object:
**/Shares/Water Buffalo Lodge/secrethandshake.txt**

The current owner of the object is the User **fred** in domain **Burlington** and the authentication service **DefaultAuthService**.

The following users and groups have been added to the Access Control List (ACL) for this object. To modify a user or group's permissions, place a check mark next to the user or group and click **Edit All Checked**. To add a user who is in the current grid domain, click **Add User to ACL**. To add a group that is in the current domain, click **Add Group to ACL**. To add a user or group that is in a connected domain, click **Add Via Interconnect ID**.

| Select | Name | Type | Domain | Auth Service | Read | Write | Execute | Delete |
|---|---|---|---|---|---|---|---|---|
| ☐ | fred | User | Burlington | DefaultAuthService | Allow | Allow | Allow | Allow |
| ☐ | WaterBuffaloes | Group | Burlington | DefaultAuthService | Allow | Unset | Allow | Unset |
| ☐ | Administrator | User | Burlington | DefaultAuthService | Allow | Allow | Allow | Allow |
| ☐ | barney | User | Burlington | DefaultAuthService | Deny | Unset | Deny | Unset |

Manage Access Control Lists

| | |
|---|---|
| Edit All Checked | Configure permissions for each user or group in the ACL for an object |
| Add User to ACL | Add a user to the ACL for an object |
| Add Group to ACL | Add a group to the ACL for an object |
| Add Via Interconnect ID | Upload an interconnect ID to add a user or group to the ACL for an object |

Notice that the ACL includes three users—fred, Administrator, and barney—and one group, WaterBuffaloes. The users fred and barney are both members of the WaterBuffaloes group; they are listed separately because their permissions are different from those of the group.

## Ownership

The user fred, listed first in the sample ACL, owns the file. Only the owner of an object or a member of the Administrators group can change the object's ACL.

**Note** Deny permissions set for the owner of an object or for members of the Administrators group are ignored—that is, you can set an ACL to deny read, write, execute, or delete permission, but the owner or administrator will still be able to perform those actions. The UI lets an owner set deny permissions for herself in case she wants to have them in effect after she sets a new owner for the object.

In Avaki, object owners can be either users or groups. However, ownership by groups is only partially supported by NFS. An Avaki group that owns an object cannot be properly mapped to a local user over NFS. (The group will be mapped to a default UID if one is configured for the object's authentication service.) Consequently, from an NFS client, it is not possible to change permissions (**chmod**), change ownership (**chown**), or change group (**chgrp**) on an object whose owner is a group. We recommend that you avoid assigning Avaki groups as owners of files, directories, and other objects that will be accessed via NFS—that is, via an Avaki data grid access server (DGAS).

Over CIFS, you can neither view nor modify object ownership information. (Nor can you change permissions, change group, or modify attributes.)

When you create a file through a DGAS, your UID and GID are mapped to a user and group on the file's home machine if mappings have been configured in the DGAS. That user and group become the owner and group owner of the file, and if the mappings are set up correctly, you can view this information through the DGAS. However, nothing else in Avaki supports the notion of group ownership. For example, an Avaki share does not import group ownership when it pulls a new file into the data catalog, and neither the web UI nor the CLI lets you view or manipulate group ownership of shared files.

## Permission settings

For each action (read, write, execute, delete), an ACL can include one of these permissions:

- Unset
  Indicates that this value has not been set. When a permission value is Unset, the user's or group's permission for this action (read, write, execute, or delete) may depend on other permissions (group or individual) for this object. For example, if a user's own account has a permission of Unset for reading a file but the user belongs to a group that is allowed to read the file, the user is allowed to read the file. If the user's account has a permission of Unset for reading a file and no group permissions apply, the user is not allowed to read the file.
- Deny
  The user or group may not perform this action (read, write, execute, or delete) on the object. When user and group permissions on an object conflict, Deny generally

wins—but see "Interpreting permissions," below, for details. Note that it is not possible to deny permission to a group, except in the case of the "everyone" group.

- Allow
  The user or group may perform this action (read, write, execute, or delete) on the object unless permission is denied elsewhere. (For example, if a user belongs to a group that is allowed to read a file, but the user himself is denied permission to read that file, the user is not able to read the file.)

## Interpreting permissions

Some permissions take precedence over others. In cases where two or more permissions disagree (for example, a user belongs to a group that is allowed to read a file but the user herself is denied permission to read that file), permissions are interpreted as follows:

1. The owner of an object or a member of the Administrators group can do anything with the object. Deny permissions for the owner of an object or an administrator have no effect.
2. Permissions for the "everyone" group take precedence over all other permissions. For example, if "everyone" is allowed to read an object, denying read permission for an individual user has no effect. Conversely, if "everyone" is denied permission to read an object, allowing a user or group has no effect.

**Note** The "everyone" group is the only group for which deny permissions can be set.

3. Deny takes precedence over Allow and Unset. (This allows you to use groups effectively by granting a permission to a group, but denying the same permission to a few members of the group.)
4. User permissions take precedence over group permissions.
5. Permissions for groups other than everyone are effective only when not trumped by any of the preceding rules.

## Permissions in directories and categories

**Using categories to solve access problems.** To access an Avaki object, a user needs permissions not only on the object itself, but on all parent objects in the path to the target object. For example, to read the file Dinosaurs.doc in the Avaki directory /Shares/pets, you must have read permission on the root directory /, /Shares, /Shares/pets, and /Shares/pets/Dinosaurs.doc. This rule applies to categories as well as Avaki directories.

In some cases, a user might need access to a file (or data service or other object) that resides in an Avaki directory to which that user should not have access. For example, suppose you have a /Shares/HR directory that contains both private information about employees and a public list of telephone numbers. To make the phone list available to users outside the HR group, you can create a category for it such as /Categories/PublicInfo. Set the permissions so that members of DomainUsers are allowed to read /Categories, /Categories/PublicInfo, and /Categories/PublicInfo/phonelist.xls. This arrangement allows all Avaki users in the current domain to read the phone list, no matter how restrictive the permissions are on the file's home directory, /Shares/HR.

**Hiding objects in directories and categories.** If a user has permission to read a directory or a category, she can list the names of files, subdirectories, and other objects in that directory or category—even those objects to which she has been specifically denied read permission. If you need to hide the existence of an object, it is not sufficient to set deny permissions on the object itself. You must also deny read permission to the directory in which the object resides, to any directories into which the object is linked, and to any categories to which the object has been added.

## Permissions on new files

A newly-created file in an Avaki share can have different permissions depending on how it was created. Consider two examples, each involving a new file in an Avaki share; the share appears in the data catalog as /Shares/plans. The physical files in the share reside on a machine called Granite.

**Example 1.** First, a user called Wilma creates a file in Granite's local file system using an OS tool such as a copy or touch command. This new file appears in the data catalog the next time the /Shares/plans share is rehashed. In the data catalog, the file inherits the Avaki ACLs of its parent directory, regardless of the permissions it has in the local file system. Thus Wilma will not be the owner of the file in the Avaki ACL unless she is the owner of the parent Avaki directory.

**Example 2.** Consider next a file that's created using Avaki—in an NFS-mounted grid directory via a data grid access server (DGAS). Like the file in the first example, this file inherits the Avaki ACLs of its parent directory. However, in addition to the inherited ACLs, the NFS-created file will have ACLs for the grid user and group to which the creator of the file (logged in to an NFS client) was mapped. The mapped grid user becomes the owner of the file in the Avaki ACL. In this example, the Avaki ACLs of the parent grid directory are:

```
/Shares/plans
owner: barney
ACLs:
barney RWXD
DomainUsers RWXD
```

Suppose a Unix user called Fflintstone is mapped to a grid user called Fred and to a grid group called Waterbuffaloes. Fflintstone goes to /Shares/plans, which has been mounted on his NFS client, and creates a new file:

```
Fflintstone> cd /mnt/avaki/Shares/plans
Fflintstone> touch myfile
```

(Avaki ACLs allow Fflintstone to do this because mapped Avaki user Fred is a member of DomainUsers.)

The new file, myfile, will have not only the permissions inherited from /Shares/plans, but also permissions for Fred and Waterbuffaloes, to which Fflintstone is mapped. The ACL for myfile looks like this:

```
/Shares/plans/myfile
owner: Fred
ACLs:
barney RWXD
DomainUsers RWXD
Fred RW-D
Waterbuffaloes R--D
Everyone R---
```

For a file created by a mapped user via a DGAS, it is not possible to change the inherited Avaki ACLs by changing file permissions in the operating system. Operating system tools (such as **chmod**) affect only permissions for users and groups that exist in or map to users and groups in the local file system—in this example, Fred and Waterbuffaloes. If you want to change the permissions for barney or DomainUsers, you have to make the change using the Avaki web UI or CLI—changes to barney's or DomainUser's permissions in the operating system, SOAP, or Avaki Studio will have no effect.

## Permissions on cached objects: using groups

A cache entry—which can contain a file, directory, or the results of a database operation or data service—includes a copy of the ACL for the object at the time it was cached. If you add a user to the object's ACL after the data is cached, the new user won't get access until the entry expires out of the cache or an administrator invalidates it. (For uncached objects, adding a user to the ACL grants immediate access.)

To avoid this problem, we suggest setting permissions for cached objects on a group basis. When you add a user to a group that already has access to a cached object, the new user can access the object's cached copies immediately.

Chapter 5

# Caching

Each Avaki grid server contains a cache service that can be used to store database results, application data, or files that are frequently accessed by users. Avaki uses caching to accomplish the following goals:

- insulate production data sources from haphazard access;
- maintain good performance for users and applications;
- refresh data in a granular way based on business need; and
- ensure maximum data availability.

Caching makes remote data access practical by limiting the number of times a data request requires immediate communication with the original data source. Avaki provides a variety of caching options and features to meet diverse performance requirements. You can specify a different caching strategy for each data item, and the various caching options can be used separately or in combination.

- **Local caching** enables caching of frequently requested results near the data source. This reduces the load on the back-end data source.
- **Remote caching** caches data close to the users or applications that will use it. This cuts down on network congestion and dramatically speeds up application performance, because remote data calls are eliminated. Caches can be prepopulated and updated during off hours when network load is low, and cache configurations can be established that ensure high availability when a network is congested or unavailable for some reason.

**Note** The terms “local” and “remote” in *local caching* and *remote caching* refer to the proximity of the cache service to the data source.

Cache update frequency can be specified for a given data item. To protect production databases from unexpected load, database administrators can schedule how often data services should be rerun and cached. Cached data can also expire after a set time period, forcing a refresh of the data on the next request.

In this chapter:

**For more information on caching.** This chapter provides an overview of the caching performed by the cache services on grid servers. More information is available:

- For instructions on configuring Avaki cache services, see the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*.
- Caching in Avaki is not restricted to the cache services on grid servers; caching is also performed on data grid access servers (DGASes). A DGAS uses its own internal caches to store copies of the files and directories it serves to NFS and CIFS clients. The DGAS caches may get their data directly from the source file systems or from the cache service associated with the DGAS. For information about configuring DGAS internal caches, see the *Sybase Avaki EII Administration Guide*.

## Using cache services to improve performance

You can use caching to make the contents of a directory or the results of database operations and data services available on demand. Caching is an important component in staging and delivering provisioned/processed data efficiently to users and applications. This section discusses how cache services can be used to protect back-end data sources and network resources and reduce latency for data access operations.

Files and directories can be *pinned* explicitly in a cache (that is, marked for caching) or fetched on demand. The results of data services and database operations can be cached on demand. They can also be scheduled to run and have their results stored in one or more caches.

Ultimately, the benefit of caching data is performance; the aspect of performance that you want to optimize determines how you set up your caching services.

If you have a widely distributed Avaki deployment, you will probably want to cache files and the results of data services and database operations close to their consuming users and applications. We refer to this as remote caching; it reduces latency caused by network delays and can improve the overall performance of your corporate network—either because repeated requests for on-demand data can be addressed locally or because you stage the results by scheduling a database operation or data service to run at particular times.

Many database administrators will also want to manage the load on their databases. They can do this using local caching. By scheduling database operation results to be cached in the cache service that is on the same grid server as the database connector, or by allowing such database operations to be cached on demand in that cache service, you can prevent queries from hitting an operational data store each time a database operation is executed.

The results of complex data processing computations in data services can also be cached, either in an on-demand fashion or following a schedule.

## *Configuring clients and Avaki servers to use cache services*

Avaki clients (such as command line clients) and JDBC applications that use the Avaki JDBC driver can be configured to take advantage of cache services. To enable them to take advantage of remote caching, you must set the cache service that they will use. If you are using only local caching for database operations and data services, you do not need to perform any special configuration steps.

For information about configuring remote caching for JDBC programs, see the *Sybase Avaki EII API Guide*.

For information about configuring remote caching for Avaki command clients, see the *Sybase Avaki EII Command Reference* and the *Sybase Avaki EII Administration Guide*.

Avaki grid servers and data grid access servers (DGASes) can also be configured to use a particular cache service in a remote-caching configuration. To learn how to configure remote caching for a grid server or DGAS, see the *Sybase Avaki EII Administration Guide*.

## *Configuring caching for files*

There are two modes in which an Avaki cache service can cache the file data and metadata for shared files: on-demand caching and scheduled (pinned) caching.

### On-demand caching

The cache service coherence window controls cache coherence for files that are cached on demand. This value can be overridden on a per-file basis by coherence windows set on individual files (either as an attribute on the back-end file, which dictates its coherence for all cache services, or during the tagging process, which dictates the coherence for that file on that particular cache service).

The cache coherence window determines how frequently a cache service will consult a file to see if it has changed. For files, this is determined by consulting the modification time (mtime) for the file or directory.

In order for a cache service to cache files on demand, you must specifically identify the file or files that should be cached on demand. Thus, a cache service will not automatically cache every single file that it may be asked for; it will cache only those that have been marked for on-demand caching. You can mark individual files or directory hierarchies of files for on-demand caching. When you mark a directory (and potentially its subdirectories) for on-demand caching, the cache service will track the directories and their contents, but not cache the file content. When a client of the cache service—a DGAS, command line interface, or SOAP client—asks for file content, the cache service pulls down and caches that file content in blocks as needed.

## Pinning files in a cache

The alternative to on-demand caching is to pin files in a cache service for scheduled caching. As with on-demand caching, pinning involves marking individual files or directory hierarchies of files to be cached. If you specify a schedule, the cache service pulls down file content during the pinning process and actively keeps the file content refreshed in the cache according to the schedule. If you do not specify a schedule, the cache service uses a schedule based on the applicable coherence window for that file. When the time specified for the coherence window elapses, the cache service consults the modification time for files and directories and syncs down any changes.

## Permissions and access control

Files and directories are pinned using the identity of the individual who marked them. Like data services and database operations, the cache service caches access control information and performs local access control checks. Access control lists (ACLs) and other metadata are refreshed according to a cache coherence window interval.

# Configuring caching for database operations and data services

There are two modes in which an Avaki cache service can cache the results-data and metadata for database operations and data services: on-demand caching and scheduled caching.

## On-demand caching

On-demand caching is suitable for Avaki objects that are rarely accessed or that change at irregular intervals. When you tag a database operation or a data service for on-demand caching, the results are cached only when the database operation or data service is executed.

Cache coherence in Avaki is maintained via coherence window mechanisms. For database operations and data services, you can use the data expiration interval to control data freshness. The data expiration interval is a settable property for data services and database operations. You can set the value of this property using either the web user interface or the command line interface.

The data expiration interval determines whether data is cached and, if so, when the data expires from the cache. You can specify that the data should never expire, or you can specify the interval before the data expires.

The default data expiration interval for a database operation or data service is no caching. You can change this data expiration interval when the database operation or data service is created or at any other time by using the view/edit option when viewing a list of database operations or data services. When the database operation or data service is executed, any results will be held in the cache for a period equal to the data expiration interval. Caching is keyed by database operation or data service name and parameter values, so any invocations of the database operation or data service with the same parameter values will be read from the cache during that time. Note that this means that multiple results for a particular database operation or data service may be in the cache at any given time.

When a database operation or data service has cached on-demand results-data, other metadata information about the database operation or data service may be cached as well. This includes security information and attributes. If these values are changed while data is being cached, we recommend that you invalidate the database operation or data service in the cache service to force a reset of all cached data for that database operation or data service. Cached data can be invalidated on the View Cache Service UI page.

## Scheduled caching

Scheduling is useful if you want the database operation or data service to run at a particular interval (minutes, hours, days, and so on). When a database operation or data service is scheduled, the results-data will stay in the cache until the scheduled execution takes place. During this time, the cache service assumes the content to be fresh and does not check with the source to see if the content has expired. The scheduling for a database operation or data service applies only to the cache service on which the schedule was created. The data expiration interval will be ignored (in the specific cache) for the database operation or data service when it has been scheduled.

## Remote/local caching interactions

The simplest caching behavior is when you are using only a local cache and you invoke a database operation or data service that has a data expiration interval higher than zero. The first time you invoke such a database operation or data service, a cache entry will be created. For the duration of the data expiration interval, data will be read from the cache.

With multiple caches, a user must be a little more careful about applying a strategy. The simplest scenario is to use on-demand caching only, by controlling the data expiration interval. In this case, the local cache and the remote cache will treat the data in the same manner. The data will expire in both caches at the same time and will be always fresh relative to the data expiration interval.

If you are implementing scheduling, you should understand which cache is taking the direct user requests before deciding on a strategy. The scheduling should be done in the cache that is taking the user requests—the remote cache, typically. If both local and remote caches are taking user requests, then scheduling in both caches may be useful. If you are using a combination of scheduling and on-demand caching then scheduling should be in the remote cache only. If not, the remote cache may not work correctly because it can receive repeated stale data from the scheduled database operation or data service in the local cache, and will never actually cache it.

## Permissions and access control

By default, you need to be a member of the Administrators group in order to configure cache services.

When you schedule database operations or data services in a particular cache, the identity that the cache uses to invoke the database operation or data service is the identity that was used to create the schedule. So if you are scheduling a database operation or data service, you must make sure that you have execute permission on it.

When a database operation or data service is accessed via a cache service, the cache service performs the access control check for the user who is invoking the service. In order to do this, the cache maintains a cached version of the object's access control list (ACL). The cache service will refresh the ACL either on a schedule or according to the cache coherence window interval.

For information about configuring caching for database operations and data services, see the *Sybase Avaki EII Provisioning and Advanced Data Integration Guide*.

# 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*