InterSystems CACHÉ®

Introduction to Caché

Version 2009.1
30 June 2009

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Introduction to Caché
Caché Version 2009.1 30 June 2009
Copyright © 2009 InterSystems Corporation
All rights reserved.

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at www.w3c.org. The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Customer Support

Tel:	+1 617 621-0700
Fax:	+1 617 374-9391
Email:	support@InterSystems.com

Table of Contents

List of Figures

List of Tables

About This Book

This book describes Caché, a high-performance database that combines an object database, high-performance SQL, and powerful multidimensional data access.

This book addresses the following topics:

- What Is Caché?
- The Caché Database Engine
- Objects, SQL, and the Unified Data Architecture
- Caché Advanced Security
- Front-end Development
- Connectivity
- Development Tools and Database Utilities

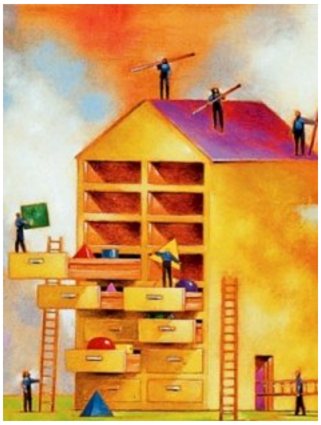
For a detailed outline, see the Table of Contents.

For information about fundamental Caché topics, see the following documents:

- *Using Caché Objects*
- *Using Caché SQL*
- *The Caché System Administration Guide*
- *The Caché Security Administration Guide*

For general information, see *Using InterSystems Documentation*.

1 What Is Caché?



Welcome to Caché, a high-performance object database.

This introduction provides an overview of the major components and technologies that make up Caché. These components include:

- A powerful, multidimensional transaction engine that includes the ability to create distributed databases.
- A unified data architecture uniting the power of objects with high performance SQL.
- A suite of technologies and tools that provide rapid development for database and web applications.
- Native, object-based XML and Web Services support.
- Automatic interoperability via Java, EJB, JDBC, ActiveX, .NET, C++, ODBC, XML, SOAP, Perl, Python, and more.

Please read the rest of this document to learn more about Caché. For more details on a specific topic, refer to one of the other books available from the online documentation home page. In addition, Caché includes a number of online tutorials on various development and system administration topics.

1.1 A Unique Architecture

Caché derives much of its power from its unique architecture. At the core, the Caché database engine provides the complete set of services — including data storage, concurrency management, transactions, and process management — needed to build complex database management systems. You can think of the Caché engine as a powerful database toolkit. Using this toolkit, Caché implements a complete object and relational database management system.

The benefits of this architecture are manifold:

- The object and relational database systems talk directly to the database engine for extremely efficient operation; there is no object-relational middleware or SQL-to-object bridge technology.
- The logical separation of the database from its physical implementation makes it possible to radically reconfigure application deployments with no change to application logic.
- Because the database engine interface is open, you can make direct use of its features where needed. This can range from building your own customized database management system to adding targeted optimizations to performance critical applications.
- A platform for the future: The Caché architecture makes future database engine enhancements possible without impact on existing applications. For example, Caché v4.1 introduced a brand-new physical data structure, with dramatically improved scalability and performance, that not only required no change to existing applications but also required no change to the Caché object or relational systems. As new technologies emerge, such as XML, Caché can add support for them as native, high-performance components with little impact to existing applications.

1.2 A High-performance Object Database — with Relational Access

Caché is designed to transcend the limitations of the relational model while providing an evolutionary upgrade path for the thousands of existing relational database applications as well as support for the many SQL-based reporting tools on the market.

In addition to being a high-performance object database, Caché is also a full-featured relational database. All the data within a Caché database is available as true relational tables and can be queried and modified using standard SQL via ODBC, JDBC, or object methods. Because of the power of the underlying Caché database engine, we believe that Caché is the fastest, most reliable, and most scalable relational database available today.

What's more, Caché offers a range of features that go beyond the limits of relational databases, while still supporting a standard relational view of data. These features include:

- The ability to model data as objects (each with an automatically created and synchronized native relational representation) while eliminating both the impedance mismatch between databases and object-oriented application environments as well as reducing the complexity of relational modeling.
- A simpler, object-based concurrency model.
- User-defined data types.
- The ability to take advantage of methods and inheritance, including polymorphism, within the database engine.
- Object-extensions for SQL to handle object identity and relationships.
- The ability to intermix SQL and object-based access within a single application, using each for what they are best suited.
- Control over the physical layout and clustering used to store data in order to ensure the maximum performance for applications.

While most databases with both object and relational access provide one form of access on top of the other, the SQL and object aspects of Caché both go directly to the data — so that users enjoy the performance benefit of either approach.

1.3 Caché in Action

Caché is used around the world for a wide variety of applications ranging from single-user embedded systems to enterprise-wide multiserver installations with tens of thousands of concurrent users.

A small sample of applications built with Caché includes:

- As the application platform for a large health-care network running hundreds of patient-critical applications. The network includes a set of Caché systems acting as data and application servers and has over 30,000 client machines.
- As the data server for a Java-based enterprise messaging system for large financial institutions. Caché was chosen both for its performance and its ability to carry out customized tasks not possible within a traditional relational database.

- As an SQL-based OLTP (online transaction processing) system for a large government organization with over 1400 concurrent users. Caché was a drop-in (no application changes) replacement when other relational products failed to perform.
- As an object database and Web application framework for an online educational system used by a leading technical university. Caché was chosen for its rapid development (the application had to be built in three months), its object capabilities, as well as its ability to scale without application reworking.
- As an object database used to track real-time position and velocity of professional athletes during a world championship. Caché was chosen for its performance (compared with the leading object and relational databases) and its native C++ interface.
- As a distributed SQL data engine for a major Web site with millions of users. This site uses a set of cost-effective Linux-based servers and uses the Caché distributed data management to provide a scalable, personalized site with no middleware or Web caching infrastructure. The hardware costs of this system (four off-the-shelf Linux machines) were less than 10% of those quoted by a “leading database for internet applications” .

1.4 Accessibility — Section 508

InterSystems believes that its products and services can be used by individuals regardless of differences in physical ability. We are committed to compliance with section 508 of the Rehabilitation Act of 1973 (29 U.S.C. 794d), as amended by Congress in 1998. We welcome and encourage suggestions that improve the accessibility and usability of our offerings; please contact us if you have contributions or questions.

1.5 Contacting InterSystems

Support

For support questions about any InterSystems products, please contact the InterSystems Worldwide Support Center:

- Telephone: +1 617 621-0700
- Fax: +1 617 374-9391
- Email: `support@intersystems.com`
- Web: http://www.intersystems.com

Section 508 — Accessibility

For questions or suggestions regarding the Rehabilitation Act (29 USC 794d) section 508:

- Telephone: +1 617 621-0700
- Fax: +1 617 374-9391
- Email: `section508@intersystems.com`
- Web: http://www.intersystems.com

2 The Caché Database Engine

At the heart of Caché lies the Caché Database Engine. The database engine is highly optimized for performance, concurrency, scalability, and reliability. There is a high degree of platform-specific optimization to attain maximum performance on each supported platform.

Caché is a full-featured database system; it includes all the features needed for running mission-critical applications (including journaling, backup and recovery, and system administration tools). To help reduce operating costs, Caché is designed to require significantly less database administration than other database products. The majority of deployed Caché systems have no database administrators.

The major features of the database engine are described in the following sections.

2.1 Transactional Multidimensional Storage

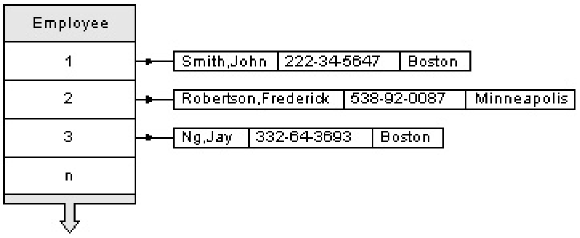
All data within Caché is stored within sparse, multidimensional arrays. Unlike the multidimensional arrays used by typical OLAP (online analytic processing) products, Caché supports transaction processing operations (inserts, updates, locking, transactions) within its multidimensional structures. Also, unlike most OLAP engines, these multidimensional structures are not limited in size to available memory. Instead, Caché includes a sophisticated, efficient data cache.

Because Caché data is of inherently variable length and is stored in sparse arrays, Caché often requires less than half of the space needed by a relational database. In addition to reducing disk requirements, compact data storage enhances performance because more data can be read or written with a single I/O operation, and data can be cached more efficiently.

2.1.1 Objects and Multidimensional Storage

Caché object technology uses multidimensional storage as the foundation of its object persistence technology. For example, suppose you have a simple Employee class that represents employee data — say an employee's name, ID number, and location. Instances of Employee object could be stored in a multidimensional array like the one pictured here.

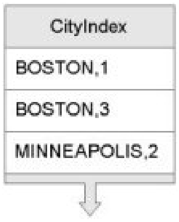
Storage of Persistent Objects



In this case, the array is subscripted by an object identifier value and the data for each instance is stored compactly within the nodes of the array. Caché automatically creates the optimal storage structure for persistent classes.

If a cross-index, say on the Location property, is needed, the persistence engine would use a different multidimensional array, similar to the one pictured below, to associate location values with corresponding object identifiers (typically a two–dimensional structure is used for this purpose).

Storage of Indices



Such indices are automatically maintained as changes are made to the database. The Caché SQL engine automatically uses such an index to satisfy queries for finding Employees by location.

2.1.2 Flexibility

Multidimensional arrays give applications a great degree of flexibility in how they store their data. For example, a set of closely related objects, say an Invoice object and its corresponding LineItem objects, can easily be configured so that the LineItem objects are physically clustered with a Invoice object for highly efficient access.

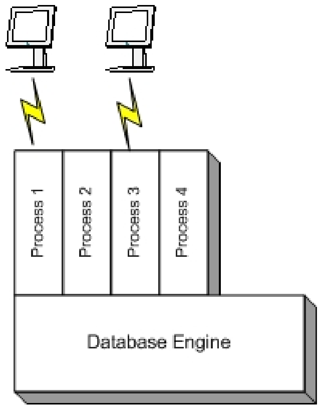
Using a unique feature known as "subscript mapping," you can specify how the data within one or more arrays is mapped to a physical database file. Such mapping is a database administration task and requires no change to class/table definitions or application logic. Moreover, mapping can be done within a specific sparse array; you can map one range of values to one physical location while mapping another to another file, disk drive, or even to another database server. This makes it possible to reconfigure Caché applications (such as for scaling) with little effort.

The flexibility of transactional multidimensional storage gives Caché a significant advantage over the two–dimensional structure used by traditional relational databases: it is this flexibility that allows Caché to be a high-performance SQL, object, and XML database without compromise. It also means that Caché applications are better prepared for future changes in technology.

2.2 Process Management

Caché provides the ability to execute database operations as well as any degree of business logic within Caché processes.

Process Management



A process is an instance of a Caché virtual machine running on a Caché server. A typical Caché server can run thousands of simultaneous processes depending on hardware and operating system. Each process has direct, efficient access to the multidimensional storage system.

The Caché virtual machine executes instructions, referred to as "P-code" , that is highly optimized for the database, I/O, and logic operations typically required by transaction processing and data warehousing applications. Virtual machine code can be created in the following ways:

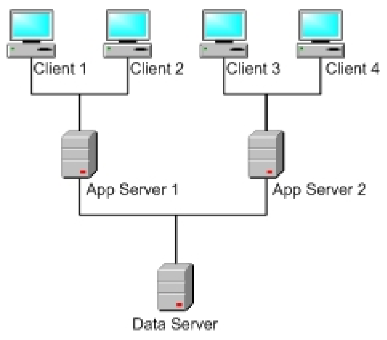
- SQL — SQL queries submitted to Caché are processed by the Caché SQL optimizer which, in turn, converts them to efficient, executable P-code (making use of any indices that may be present).

- Object Behavior — The Caché Objects technology provides a high degree of server-side object behavior (such as object persistence) by automatically generating executable P-code (using method generators — object methods that can generate code according to pre-determined rules).
- Caché ObjectScript — Applications can include any logic that they wish to execute within a Caché data or application server using the Caché ObjectScript scripting language. Such code can take the form of object methods (analogous but much more powerful than stored procedures within the relational world as they can make full use of object-oriented features) or complete “routines” (small programs that run within a Caché server).
- Caché Basic — Object methods can also be implemented using the Basic programming language. Caché includes a powerful, object-based version of the popular Basic programming language that is familiar to a large portion of the world’s software developers. Caché Basic runs on every supported platform and is completely interoperable with Caché ObjectScript.

2.3 Distributed Data Management

One of the most powerful features of Caché is its ability to link servers together to form a distributed data network. In such a network, machines that primarily serve data are known as Data Servers while those that mainly host processes, but little to no data, are known as Application Servers.

Enterprise Cache Protocol



Servers can share data (as well as locks) using the Caché Enterprise Cache Protocol (ECP). ECP is effective because data is transported in packages. When information is requested across the network, the reply data package includes the desired data, and additional, related data as well. The natural data relationships inherent to objects and the Caché multidimensional data model make it possible to identify and include information that is related to the originally requested data. This “associated” information is cached locally either at the client or on the application server. Usually, subsequent requests for data can be satisfied from a local cache, thus avoiding additional trips across the network. If the client changes any data, only the updates are propagated back to the database server.

ECP makes it possible for applications to support a wide variety of runtime configurations including multi-tier and peer-to-peer.

2.4 Journal Management

To provide database integrity and reliability, Caché includes a number of journaling subsystems that keep track of physical and logical database updates. The journal management technology is also used to provide transaction support (a journal is used to perform transaction rollback operations) as well as database shadowing (a journal is used to synchronize a shadow server with a primary data server). As with the rest of the system, Caché lets you configure its journaling system to meet your needs.

2.5 Lock Management

To support concurrent database access, Caché includes a powerful Lock Management System.

In systems with thousands of users, reducing conflicts between competing processes is critical to providing high performance. One of the biggest conflicts is between transactions wishing to access the same data. Caché lock management offers the following features to alleviate such conflicts:

- Atomic Operations — To eliminate typical performance hot spots, Caché supports a number of atomic operations, that is with no need for application level locks. An example of this is the ability to atomically allocate unique values for object/row identity (a common bottleneck in relational applications).
- Logical Locks — Caché does not lock entire pages of data while performing updates. Because most transactions require frequent access or changes to small quantities of data, Caché supports granular logical locks that can be taken out on a per-object (row) basis.
- Distributed Locks — in distributed database configurations, Caché automatically supports distributed locks.

2.6 Device Management

Caché provides portable support for myriad devices (such as files, TCP/IP, printers) making it possible for Caché applications to interoperate with a host of other technologies. The interconnectivity options available with Caché (including CSP, ODBC, SOAP, and Java) are built on top of this underlying support.

2.7 Portability

Caché runs on, and is optimized for, a wide variety of hardware platforms and operating systems, as documented in the manual *Supported Platforms.*

You can easily port applications developed with Caché as well as data from one platform to another. This can be as easy as installing Caché on the new platform and moving the database files to new system. When moving between some systems, you may need to run an in-place data conversion utility (to convert one endian representation to another).

2.8 Deployment Options

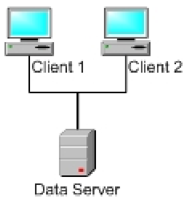
Caché supports a variety of different runtime configurations giving you maximum flexibility when you deploy your applications. You can switch between different deployment options by changing Caché system settings; typically there is no need to change your application logic.

Some basic deployment options are listed below.

2.8.1 Basic Client/Server Configuration

In the simplest client/server configuration, a single Caché data server services a number of clients (from one to many thousands, depending on the application and platform).

Client/Server Configuration



The client systems can be any of the following:

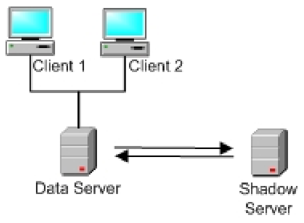
- Stand-alone desktop systems running a client application that connects via a client/server protocol (such as ODBC, ActiveX, JDBC, Java).
- Web server processes talking to Caché via CSP (Caché Server Pages), SOAP, or some other connectivity option (such as ODBC, JDBC). Each Web server process may then service a number of browser-based or machine-to-machine sessions.

- Middleware processes (such as an Enterprise Java Bean application server) that connect to Caché via ODBC, JDBC, etc.
- Devices, such as terminals or lab equipment, that connect to Caché using one of many supported protocols (including TELNET and TCP/IP).
- Some combination of the above.

2.8.2 Shadow Server Configuration

The Shadow Server configuration builds upon the basic client/server setup by adding one or more shadow servers. Each shadow server synchronizes itself with the data within the main data server by connecting to and monitoring its transaction journal.

Shadow Server Configuration

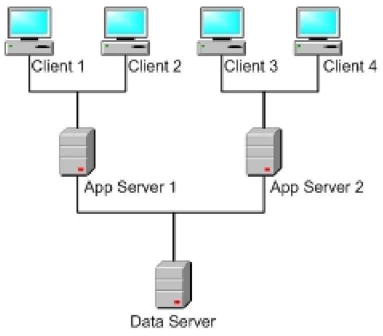


Shadow servers are typically used to service ad hoc queries, large reports, and batch processes to limit their impact on the main transaction system. They can also be used to provide fail-over systems.

2.8.3 Multi-tier Configuration

The multi-tier configuration uses the Caché distributed database technology — the Enterprise Cache Protocol (ECP) — to make it possible for a greater number of clients to connect to the system.

Multi-tier Configuration



In the simplest multi-tier setup, one or more Caché systems, acting as application servers, are placed between the central data server and the various client systems. In this case the application servers do not store any data, instead they host processes that perform work for the benefit of the client, off-loading the CPU of the data server. This type of configuration scales best for applications that exhibit good "locality of reference," that is most transactions involve reasonably related data so that locking across application servers is limited. Such applications, as well as those with a fair amount of read access (like most typical Web applications), work extremely well in this model.

More complex configurations, with multiple data servers as well as data stored on application server machines, are also possible.

Typically applications use the multi-tier configuration for scaling as well as for providing high-availability (with applications servers serving as hot standby systems).

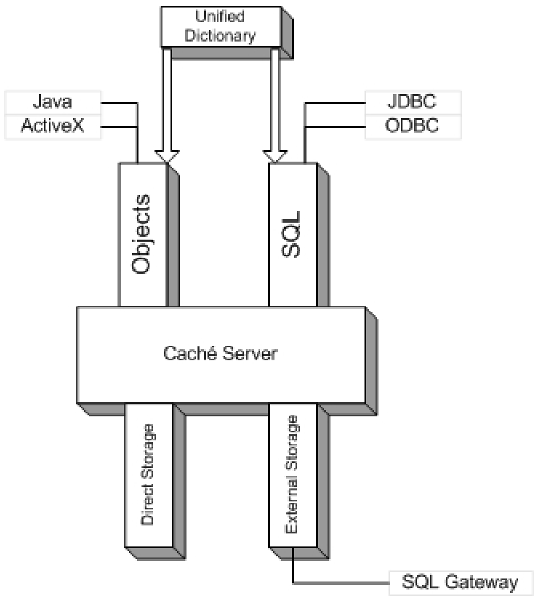
3 Objects, SQL, and the Unified Data Architecture

A powerful and unique feature of Caché is its unique Unified Data Architecture that provides simultaneous, high-performance object and relational access to data stored within Caché.

3.1 Unified Data Dictionary

Within Caché, you can model your application components as objects. Objects are organized by classes which define the data (properties) and behavior (methods) of the object.

Unified Data Architecture



The meta-information, or definition, of each class is stored within a common repository referred to as the Caché class dictionary. The class dictionary is itself an object database, stored within Caché, whose contents can be accessed using objects. The class dictionary, by means of a class compiler, defines the storage structure needed by persistent objects and converts class definitions into parallel sets of executable code that provide both the object and relational access to this storage structure. By means of this architecture, the object and relational code paths are efficient and automatically synchronized with one another.

Class definitions can be added to the class dictionary in a number of ways:

- Interactively, using the Caché Studio development environment.
- Relationally, using DDL. Caché accepts standard SQL DDL statements and automatically creates corresponding class and table definitions.
- Textually, using XML. Caché supports an external, XML representation of class definitions. Typically this is used for source code management, deployment, automatic code generation, and interoperation with other tools.
- Programmatically, using objects. Using the Caché set of class definition objects, you can create programs that communicate directly with the class dictionary and create new classes at application runtime.
- Using an XML Schema Wizard, included within Caché Studio, that can create class definitions from most XML schema files.

3.1.1 Flexible Storage

The Caché object model differs from those of programming languages in that in addition to properties and methods, you can specify storage-related behavior such as indices, constraints, and storage structure.

The storage structure used by persistent objects is independent of the logical definition of a class and is quite flexible: developers can use the default structures provided by the class compiler or they can tune the structures for specific cases.

3.2 Objects

Caché includes a full-featured, next-generation object database specifically designed to meet the needs of complex, transaction oriented applications. The Caché object model includes the following features:

- Classes — You can define classes that represent the state (data) and behavior (code) of your application components. Classes are used to create instances of objects as both runtime components and as items stored within the database.
- Properties — Classes can include properties, which specify the data associated with each object instance. Properties can be simple literals (such as strings or integers), user-defined types (defined using data type classes), complex (or embedded) objects, collections, or references to other objects.
- Relationships — Classes can define how instances of objects are related to one another. The system automatically provides navigational methods for relationships as well as referential integrity within the database.
- Methods — Classes can define behavior by means of methods: executable code associated with an object. Object methods run within a Caché server process (though they can be invoked from a remote client). Object methods can be scripted using Caché ObjectScript, SQL, or they can be generated using method generators, which are code that automatically creates customized methods according to user-defined rules.
- Object persistence — Persistent objects have the ability to automatically store and retrieve themselves to a database. The persistence support includes complete database functionality including automatic transaction management, concurrency control, index maintenance, and data validation. Persistent objects are automatically visible through SQL queries.
- Inheritance — By deriving new classes from existing ones, you can reuse previously written code as well as create specialized versions of classes.
- Polymorphism — Caché supports complete object polymorphism. This means that applications can use a well-defined interface (a set of methods and properties provided by a superclass) and the system will automatically invoke the correct interface implementation based on the type of each object. This makes it much easier to develop flexible database applications.

- Swizzling (also known as "lazy loading") — Caché automatically swizzles (brings into memory from disk) any related persistent objects when they are referenced from other objects. This greatly simplifies working with complex data models.

The Caché object functionality is not a separate part of Caché; it is a central part of Caché programming and is fully integrated with relational access described elsewhere. However, for those who are interested specifically in object-oriented programming, the manual Using Caché Objects discusses Caché programming from this point of view.

3.2.1 Defining Classes

The simplest and most common way to define classes within Caché is to use the Caché Studio development environment. The Studio lets you define classes either using a simple text format within a syntax-coloring editor or by using a graphical point-and-click interface. These two views are interchangeable and are automatically synchronized.

Here is the definition of an extremely simple persistent object, Component, as seen within Caché Studio:

```
Class MyApp.Component Extends %Persistent [ClassType = persistent]
{
Property TheName As %String;
Property TheValue As %Integer;
}
```

This class is defined as a persistent class (that is, it can store itself within a database). In this case, the Caché-provided, %Persistent class (system class names start with a "%" character to distinguish them from application classes) provides all the needed persistence code via inheritance. The class belongs to the package, "MyApp". Packages group related classes together and greatly simplify development of large applications. The class defines two properties: TheName, which has a string value, and TheValue, which has an integer value.

From within Caché ObjectScript code, such as within a method, you can use this object syntax to manipulate instances of Component object:

```
 // Create a new component
 Set component = ##class(MyApp.Component).%New()
 Set component.TheName = "Widget"
 Set component.TheValue = 22

 // Save the new Component to the database
 Do component.%Save()
```

Using Basic, you can define a method to manipulate instances of the Component object:

```
' Create a new component
component = New Component()
component.TheName = "Widget"
component.TheValue = 22

' Save the new Component to the database
component.%Save()
```

At this point, a new instance of Component is stored within the database with a system-assigned unique object identifier. You can later retrieve this object by opening it (using its object identifier):

```
' Open an instance and double its value:
component = OpenId Component(id)

component.TheValue = component.TheValue * 2
component.%Save()
```

You can perform the exact same operations using native Java, C++, or other Caché client bindings. The class compiler can generate, and synchronize, any additional code required to access objects externally. For example, if you are using Caché with Java, you can specify that the class compiler automatically generate and maintain Java proxy classes that provide remote access to persistent database classes. Within a Java program you can use this object naturally:

```
// Get an instance of Component from the database
component = (MyApp.Component)MyApp.Component._open(database, new Id(id));

// Inspect some properties of this object
System.out.println("Name: " + component.getTheName());
System.out.println("Value: " + component.getTheValue());
```

3.3 SQL

Caché SQL is a full-featured relational database engine that is fully integrated with the Caché object technology. In addition to standard SQL-92 features, Caché SQL offers:

- Support for streams (known in SQL as Binary Large Objects, or BLOBS).
- Support for stored procedures (implemented as object methods).
- A set of object-based extensions.
- User-definable data types.
- Support for Transactional Bitmap Indices.

  Bitmap indices, typically used in large data warehousing and OLAP systems, offer the ability to perform high-speed searches based on complex combinations of conditions. Such bitmap indices cannot be updated in real-time, however and are typically updated as a batch process. Caché SQL supports bitmap indices that offer high-performance searching power combined with no loss in insert/update performance. This gives transaction processing applications the ability to perform data warehouse-style queries and gives data warehouse applications the ability to perform real-time updates. For more information, refer to the Caché SQL documentation.

3.3.1 The Object/Relational Connection

All components within the Caché dictionary are defined as classes. The Caché class compiler automatically projects persistent classes as relational tables. For every object feature, there is a corresponding relational equivalent, as illustrated in the following table:

Relational View of Object Features

Object Feature	Relational Equivalent
Package	Schema
Class	Table
Object instance	Row within a table
Property	Column
Relationship	Foreign key
Embedded object	Multiple columns
Method	Stored procedure
Index	Index

When Caché loads SQL DDL (Data Definition Language) statements, it uses the inverse of this projection to create classes that correspond to relational tables.

To demonstrate the object-to-relational projection, consider a simple example. Here is the definition of a simple, persistent Person class (part of a package called "MyApp") containing two properties, Name and Home:

```
Class MyApp.Person Extends %Persistent [ClassType = persistent]
{
Property Name As %String(MAXLEN=100);
Property Home As Address;
}
```

The Person class gets its persistent behavior from the %Persistent superclass provided with Caché. The Name property is defined as a simple String of up to 100 characters.

The Home property illustrates the use of complex, user-defined data types, in this case the Address class, which is defined as:

```
Class MyApp.Address Extends %SerialObject [ClassType = serial]
{
Property City As %String;
Property State As %String;
}
```

The Address class is derived from the %SerialObject superclass. This class provides the ability to serialize itself (convert itself to a single-string representation) and embed itself within another containing class (as with the Person class).

When viewed via SQL, the Person class has the following structure:

SQL View of the Person class: SELECT * FROM Person

ID	Name	Home_City	Home_State
1	Smith,John	Cambridge	MA
2	Doe,Jane	Dallas	TX

Note that the object identifier is visible as a column. In addition, the fields of the embedded Address object are projected as separate fields. These fields are given the synthetic names Home_City and Home_State and behave exactly as if they were defined as two individual fields.

3.3.2 Inheritance and SQL

Inheritance is an important feature within object-based systems and is completely lacking within relational databases. Caché SQL makes it possible to use the power of inheritance using standard relational constructs. For example, we can derive a new Employee class from the Person class used in the previous example:

```
Class MyApp.Employee Extends Person [ClassType = persistent]
{
Property Salary As %Integer(MINVAL=0,MAXVAL=100000);
}
```

This new class extends the Person class by adding an additional property, Salary.

When viewed via SQL, the Employee class has the following structure:

SQL View of the Employee class: SELECT * FROM Employee

ID	Name	Home_City	Home_State	Salary
3	Divad, Gino	Irvine	CA	22000

Notice that all of the inherited properties are available as columns. Also note that only rows that are actual instances of Employee are included. If we again ask for all Person instances:

*Revised SQL View of the Person class: SELECT * FROM Person*

ID	Name	Home_City	Home_State
1	Smith,John	Cambridge	MA
2	Doe,Jane	Dallas	TX
3	Divad, Gino	Irvine	CA

In this case, we see that all the rows are returned because every Employee is defined to be an instance of Person. In this case, however, only the properties defined by Person are displayed.

3.3.3 Object Extensions to SQL

To make it easier to use SQL within object applications, Caché includes a number of object extensions to SQL.

One of the most interesting of these extensions is ability to follow object references using the reference ("–>") operator. For example, suppose you have a Vendor class that refers to two other classes: Contact and Region. You can refer to properties of the related classes using the reference operator:

```
SELECT ID,Name,ContactInfo->Name
FROM Vendor
WHERE Vendor->Region->Name = 'Antarctica'
```

Of course, you can also express the same query using SQL JOIN syntax. The advantage of the reference operator syntax is that it is succinct and easy to understand at a glance.

4 Caché Advanced Security

Caché Advanced Security provides a simple, unified security architecture with the following features:

- It offers a strong, consistent, and high-performance security infrastructure for applications.
- It meets certification standards.
- It makes it easy for developers to build security features into applications.
- It places a minimal burden on performance and operations.
- It ensures that Caché can operate effectively as part of a secure environment and that other applications and Caché can work together well.
- It provides infrastructure for policy management and enforcement.

Caché Advanced Security is based on authentication, authorization, auditing, and database encryption. Security is discussed in detail in the *Caché Security Administration Guide*; the following sections provide an introduction.

4.1 Authentication: Establishing Identity

Authentication is how you prove to Caché that you are who you say you are. Without trustworthy authentication, authorization mechanisms are moot — one user can impersonate another and then take advantage of the fraudulently obtained privileges.

The authentication mechanisms available depend on how you are accessing Caché. Caché has a number of available authentication mechanisms:

- Kerberos — The most secure means of authentication. The Kerberos Authentication System provides mathematically proven strong authentication over a network. Kerberos is a time-tested, *trusted-third-party system* developed at the Massachusetts Institute of Technology (MIT): the Kerberos server holds all sensitive authentication information (such as passwords) and is itself kept in a physically secure location. It provides authentication over an unsecured network, and protects communications using it against sophisticated attacks. Passwords are never transmitted over the network — even encrypted.

- Operating-system–based — OS-based authentication uses the operating system's identity for each user to identify that user for Caché purposes. For Windows, OS-based authentication is available for local logins only, as Windows domain logins provide native Kerberos authentication.
- Caché login — With Caché login, Caché prompts the user for a password and compares a hash of the provided password against a value it has stored. The system manager can configure certain password criteria, such as minimum length, to ensure a desired degree of robustness in the passwords selected by users.

You can also allow all users to connect to Caché without performing any authentication. This option is appropriate for organizations with strongly protected perimeters or in which neither the application nor its data are an attractive target for attackers.

4.2 Authorization: Controlling User Access

Once a user is authenticated, the next security-related question to answer is what that person is allowed to use, view, or alter. This determination and control of access is known as *authorization*. Authorization manages the relationships of users and *resources* — entities being protected. Resources are as diverse as databases, Caché services (such as for controlling CSP access), and user-created applications.

4.2.1 Resources, Permissions, and Privileges

The primary goal of security is the protection of *resources* — information or capabilities in one form or another. With Caché, resources can be databases, services, applications, tools, and even administrative actions. The system administrator grants access to these by assigning *permissions*. Together, a resource and an associated, assigned permission are known as a *privilege*.

Resources differ from assets as follows:

- Assets are the items being protected while resources are their logical representation within the Caché security system.
- A single resource can protect multiple assets.

4.2.2 Users and Roles

Caché uses Role-Based Access Control (RBAC) for its authorization model. With this type of model, a user gains the ability to manipulate resources as follows:

1. *Resources* are associated with *permissions* to establish *privileges*.
2. *Privileges* are assigned to *roles*.
3. *Roles* have members, such as *users*.

A *user* connects to Caché to perform some set of tasks. A *role* describes a set of privileges that a user holds.

Roles provide an intermediary between users and privileges. Instead of creating as many sets of privileges as there are users, roles allow you to create sets of task-specific privileges. You can grant, alter, or remove the privileges held by a role; this automatically propagates to all the users associated with that role. Instead of managing a separate set of privileges for each and every user, you instead manage a far smaller number of roles.

For example, an application for a hospital might have roles for both a doctor making rounds (**`RoundsDoctor`**) and a doctor in the emergency room (**`ERDoctor`**), where each role would have the appropriate privileges. An individual user could be a member of just one of the two roles, or of both of them.

Caché comes with a set of predefined roles.

A user's login establishes a default set of roles for the user. Once logged in, a user may temporarily be a member of additional roles — either from a Caché application or from some part of the Caché system itself. When Caché adds new roles to a user, this is known as *escalating roles*. The removal of roles is known as *role de-escalation*.

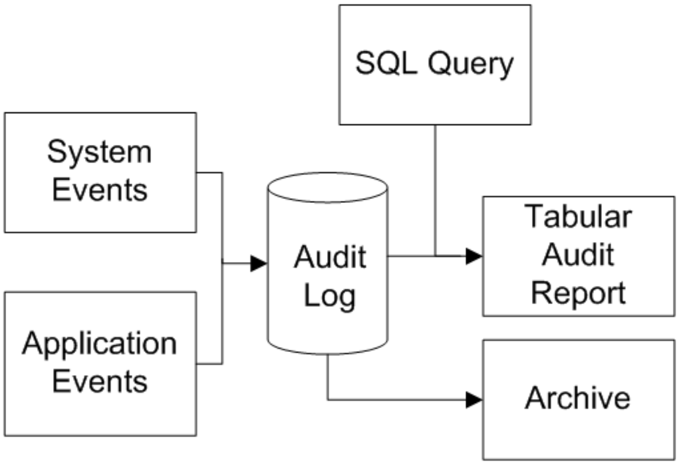
4.3 Auditing: Knowing What Happened

Auditing provides a verifiable and trustworthy trail of actions related to the system. Auditing serves multiple security functions:

- It provides proof — the proverbial "paper trail" — recording the actions of the authentication and authorization systems in Caché and its applications.
- It provides the basis for reconstructing the sequence of events after any security-related incident.
- Knowledge of its existence can serve as a deterrent for attackers (since they know they will reveal information about themselves during their attack).

The auditing facility allows you to enable logging for various system events, as well as user-defined events. Authorized users can then create reports based on this audit log, using tools that are part of Caché. Because the audit log can contain sensitive information, running an audit report itself generates an entry for the audit log. The included Caché tools support archiving the audit log and other tasks.

Caché Auditing System



Different environments require different actions if the audit log becomes full. This is why Caché makes this behavior fully configurable. If it becomes impossible to write to the audit database, you can choose to either halt Caché or overwrite the oldest existing records; the correct procedure depends upon your specific business needs.

4.4 Database Encryption: Protecting Data on Disk

Caché database encryption protects data at rest — it secures information stored on disk — by preventing unauthorized users from viewing this information. Caché implements encryption using the AES (Advanced Encryption Standard) algorithm, using 128-bit keys by default. Encryption and decryption occur when Caché writes to or reads from disk. The information handled includes the data itself, indices, bitmaps, pointers, allocation maps, and incremental backup maps.

Those experienced with encryption systems for databases may have concerns about encryption having dire effects on performance, but, with Caché, these concerns are unfounded. Encryption and decryption have been optimized, and their effects are both deterministic and small for any Caché platform; in fact, there is no added time at all for writing to the database.

5 Front-end Development

Caché provides tools to help you develop front ends, as well, so that you can create complete, rich applications.

5.1 Caché Server Pages (CSP)

The first of these tools is the Caché Server Pages (CSP) technology. **CSP** is both an architecture and toolset used to build an interactive CSP application. With CSP, you can build and deploy high-performance, highly scalable Web applications. CSP lets you dynamically generate Web pages, typically using data from a Caché database. These pages are dynamic; that is, the same page may deliver different content each time it is requested from dynamic data sources.

CSP is versatile. It can do all of the following:

- Display inventory data that changes minute by minute.
- Support Web communities with thousands of active users.
- Personalize pages based on user information stored in the Caché database.
- Customize pages based on the user data to different users, depending on their requirements and their security permissions.
- Serve HTML, XML, images, or other binary or textual data.
- Deliver fast performance, because it is tightly coupled to the high-performance Caché database.

In addition to providing rapid access to the built-in Caché database, it provides a number of features essential for Web-based database applications including session management, page authentication, and the ability to perform interactive database operations from a Web page.

For details, see the manual *Using Caché Server Pages (CSP)*.

5.2 Zen

The second, complementary tool is **Zen**. The Zen application framework provides a simple way to rapidly create complex, data-rich Web applications by assembling pre-built object components. These components automatically create standard HTML and JavaScript needed to render complex Web applications. Moreover, they provide a common object model that is shared between the user's browser and the application logic running on the server.

Zen is based on the successful Caché Server Page (CSP) and Caché Object Database technologies from InterSystems. These technologies offer a robust, scalable, and portable platform for hosting Web applications. Zen does not replace or deprecate Caché Server Pages in any way. Instead, Zen makes the development of web-based applications easier while building upon the basic features provided by CSP: performance, data access, security, localization, and configuration.

The benefits of using Zen include:

- *Rapid development* — Assembling applications from pre-built components is a proven method for creating user interfaces rapidly. Zen makes it possible to use this approach for Web-based applications.
- *Simplicity* — Zen uses a standard HTML client, so there are no additional client components required. There is no "middle" tier between server and client, so Zen applications are much easier to develop, deploy, and support.
- *Extensive library of components* — The Zen library comes with a large set of pre-built components. These include all the standard control types as well as data-aware combo boxes, tables, grids, tabs, tree controls, menus, and grouping components.
- *Object database integration* — Zen is tightly integrated with the underlying Caché object database. The Zen client and server communicate by sending objects back and forth. Underlying details such as encryption and data marshalling are handled automatically using proven Caché technology.
- *Code generation* — Much of the code and business logic is generated automatically from higher-level models, ensuring consistency and rapid development.
- *Extensibility* — You can easily modify the look and feel of Zen applications by using standard CSS style sheets; by supplying different property values for the Zen components; or by creating your own custom components.
- *Integrated use of SVG* — Zen lets you add interactive graphics to Web applications by means of a set of SVG (Scalable Vector Graphics) components. These components include pre-built charts and meters. In addition, you can create new graphical components.

- *Client independence* — Zen applications run in multiple browsers and insulate applications from the differences in behavior of these browsers.
- *Security* — Zen offers tight integration with the security model provided by the Caché object database.
- *Form handling* — The Zen framework lets you define forms that contain a variety of controls for displaying and editing of data. The framework includes extensible support for loading data into forms; validating the contents of forms; and saving form contents.
- *Page layout* — Zen includes an extensible framework for specifying the grouping and layout of components on a Web page.
- *Event management* — Zen applications can easily define how their components respond to user events. All events are handled by invoking methods that can be defined to run within the client browser or on the data server.
- *Multilingual support* — For applications that must support multiple languages, Zen includes a mechanism that tracks which titles and captions need to be localized, and *automatically* builds a database of these values. This database can be exported as XML and given to a translator for conversion into other languages. At runtime, Zen automatically displays pages in the user's preferred language.
- *Document output* — Zen includes an extensible framework for specifying the data contents and display layout of reports in XHTML or PDF format.

For details, see the manual *Using Zen*.

6 Connectivity

Caché includes a variety of technologies for efficiently and easily connecting and interoperating with almost every other software architecture.

6.1 Caché Callin and Callout Interfaces

Caché provides a **callin interface** that you can use from within C programs to execute Caché commands and evaluate Caché expressions. The callin interface permits a wide variety of applications. For example, you can use it to make Caché ObjectScript available from an integrated menu or GUI. If you gather information from an external device, such as an Automatic Teller Machine or piece of laboratory equipment, the callin interface lets you store this data in a Caché database. Although Caché currently supports only C and C++ programs, any language that uses the calling standard for that platform (OpenVMS, UNIX, Windows) can invoke the callin functions.

Caché also provides a **callout interface**, a set of functions and utilities that enable you to access external commands from within Caché. With the callout interface, you can:

- Issue operating systems commands on several platforms.
- Call routines written in other languages.
- Build your callout routines as a DLL (on Windows), a shared library (on UNIX), or a shared executable (on OpenVMS), instead of linking them statically.
- Reference your own callout modules, which might be custom dynamic link libraries (Windows), dynamic shared libraries (UNIX), or shared executable images (OpenVMS).

For details, see the manual *Using the Caché Callin and Callout Functions*.

6.2 Caché ODBC

The C-language call level interface for Caché SQL is ODBC. Unlike other database products, the **Caché ODBC driver** is a native driver — it is not built on top of any other proprietary interface. The Caché ODBC driver offers the following features:

- High performance
- Portability
- Native Unicode support
- Thread safety

You can use Caché ODBC with any tool, application, or development environment that supports ODBC.

For information on the Caché ODBC driver, see the manual *Using Caché ODBC.*

6.3 Caché JDBC

Caché includes the world’s highest performance, native JDBC driver to provide access to relational Java applications and tools. Like the Caché ODBC driver, the **Caché JDBC driver** offers high performance, portability, native Unicode support, and thread safety. You can use this driver with any tool, application, or development environment that supports JDBC.

For information, see the manual *Using Java with Caché.*

6.4 Caché XML and Web Services

Caché includes a number of ways to interoperate with XML and SOAP.

First, it provides built-in support for XML projections of any objects as well as data sets. Caché brings the power of objects to XML processing: you are not constrained to using XML as simple text files (or relational BLOBs), nor are you forced to wedge complex XML documents into relational rows and columns. Instead, you can use objects as a direct representation of XML documents and vice versa. Because Caché includes a native object database, you can use such objects directly with a database; you do not need additional complex middleware or conversion technologies. You can use XML with Caché in a variety of ways, including:

- As a standard format in messaging applications. This includes industry-standard protocols as well as more homegrown solutions.
- As a standard format for data exchange between applications and users.
- As a standard representation for external data storage. This may include traditional database records or it may include more complex content such as documentation.

You can import XML documents into Caché objects, manipulate XML objects in Caché, and export Caché objects as XML documents. You also have control over the XML schemas used by these objects, and you can export the schema as well.

The Caché support for XML makes it easy for Caché to also support SOAP and Web services. The Caché SOAP support is easy to use, efficient, and fully compatible with the SOAP specification. This support is built into Caché and does not require any complex middleware or operating system extensions. It is available on every platform supported by Caché. Using Caché SOAP support, you can do the following:

- Define and publish Web services — a collection of related methods that client applications invoke using the SOAP protocol. Such methods can be discovered and invoked by other SOAP-aware applications. Caché runs SOAP methods directly within the database, making the execution of such methods highly efficient.
- Provide a way for your applications to interoperate with other applications using the standard SOAP protocol.
- Develop highly efficient Web services (based on new development or perhaps utilizing existing application code) that can be deployed on any platform supported by Caché — you are not limited to a specific operating system or platform.

For details, see the manuals *Using XML with Caché* and *Using SOAP and Web Services with Caché*.

6.5 Caché Object Server for Java and EJB

The Caché Java Binding makes persistent Caché classes available for use within Java and EJB (Enterprise Java Bean) applications.

First, the **Caché Java binding** provides a simple, direct way to use Caché objects within a Java application. You can create Java applications that work with the Caché database in the following ways:

- The Caché Java Binding — The Caché Java binding lets Java applications work directly with objects on a Caché server. The binding automatically creates Java proxy classes for the Caché classes you specify. Each proxy class is a pure Java class, containing only standard Java code that provides your Java application with access to the properties and methods of the corresponding Caché class.

- The Caché Java binding offers complete support for object database persistence, including concurrency and transaction control. In addition, there is a sophisticated data caching scheme to minimize network traffic when the Caché server and the Java environment are located on separate machines.
- The Caché JDBC Driver (mentioned earlier in this chapter) — Caché includes a level 4 (pure Java) JDBC driver that supports the JDBC version 3.0 API. The Caché JDBC Driver provides high-performance relational access to Caché. For maximum flexibility, applications can use JDBC and the Caché Java binding at the same time.

For details, see the manual *Using Java with Caché*.

Second, the **Caché EJB binding** allows you to use Caché with an Enterprise Java Bean (EJB) application. The Caché EJB binding offers the following advantages:

- The Caché EJB binding provides the performance of hand-written persistence code without the effort of writing and maintaining it. Caché automatically generates persistence code for EJB Entity beans using the metadata stored within the Caché class dictionary.
- The Caché database is defined in terms of objects, so there is no need for cumbersome object-relational mapping within the EJB server.
- Caché EJB offers sophisticated caching to eliminate unnecessary network traffic between the EJB server and the database server.
- Caché enables faster EJB application development by eliminating much of the associated drudge work. In addition to automatically generating EJB Entity beans, Caché generates EJB deployment descriptors, scripts for deploying your beans onto the EJB server, and Java code for testing your Entity beans.
- Caché lets your EJB application use both relational and object access to the database, allowing you choose whichever is the more appropriate technique for the task at hand.

For details, see the manual *Using Caché Java with Third-party Persistence Frameworks*.

6.6 Caché Object Server for ActiveX

The **Caché Object Server for ActiveX** makes persistent Caché classes available for use within ActiveX environments such as Visual Basic or .NET. This includes the following ActiveX components:

- Caché Object Server for ActiveX — An ActiveX automation server that exposes Caché objects as ActiveX objects.
- Caché List Control — An ActiveX control written for Visual Basic that aids in the display of query results. You must provide the interface for selecting queries and query parameters for execution.

- Caché Query Control — An ActiveX control written for Visual Basic that provides a simple interface for executing queries and displaying the results. The Caché Query Control provides an interface for selecting at runtime any query that returns the ID and for specifying any query parameters.
- Caché Object Form Wizard — A Visual Basic add-in that allows you to quickly and easily create a simple form to access properties of a single Caché class.

The Caché ActiveX binding provides access to Caché from any application that supports ActiveX objects. This binding allows you to instantiate and manipulate Caché objects within Visual Basic and other client applications, and provides a transparent interface to their server-side object instantiations. The binding also includes a special set of tools for use only from Visual Basic, which are collectively called **Visual Caché**.

For details, see the manual *Using ActiveX with Caché*.

6.7 Caché Object Server for C++

The **Caché C++ binding** makes persistent Caché classes available for use within C++ applications.

- *The Caché C++ binding* — The Caché C++ binding lets C++ applications work with objects on a Caché server. The Caché Class Generator can create a C++ proxy class for any Caché class. Proxy classes contain standard C++ code that can be compiled and used within your C++ application, providing access to the properties and methods of the corresponding Caché class.

  The C++ binding offers complete support for object database persistence, including concurrency and transaction control. In addition, there is a sophisticated data caching scheme to minimize network traffic when the Caché server and C++ applications are located on separate machines.
- *Dynamic binding* — Instead of using compiled C++ proxy classes, you can work with Caché classes dynamically, at runtime. This can be useful for writing applications or tools that deal with classes in general and do not depend on particular Caché classes.
- *Light binding* — The **Light C++ binding** is a limited subset of the Caché C++ library intended primarily for loading simple data at very high speed. It combines your C++ application and the Caché Object Server into a single process, using intraprocess communications rather than TCP/IP to exchange data between them. For basic object manipulation (creating objects, opening objects by Id, updating, and deleting), it is ten to twenty times faster than the standard C++ binding.
- *The Caché ODBC driver* (introduced earlier in this chapter) — Caché includes a standard ODBC driver that offers high-performance relational access to Caché, including the ability to execute SQL queries against the database. The C++ binding provides special classes to encapsulate the complexity of ODBC. For maximum flexibility, applications can use ODBC and the Caché C++ binding at the same time.

6.8 Caché SQL Gateway

The **Caché SQL Gateway** gives Caché applications object access to third party relational databases. Using the SQL Gateway, applications can:

- Access data stored in third-party relational databases within Caché applications using objects and/or SQL queries.
- Store persistent Caché objects in external relational databases.

For details, see the chapter "Using the Caché SQL Gateway" in *Using Caché SQL*.

6.9 Caché ActiveX Gateway

The **Caché ActiveX Gateway** gives Caché applications direct access to ActiveX/COM/.NET components. By means of wrapper classes, ActiveX components are made available as instances of Caché object classes and can be used in the same manner as any other class. **Caché Activate** provides the ability to instantiate an external COM object and manipulate it as if it were a native Caché object.

Caché Activate works as follows:

1. Using the Caché Activate Wizard, you can create one or more wrapper classes. These are Caché classes that provide methods that correspond to the interface of an ActiveX component.
2. Within a Caché application, you can create an instance of an ActiveX wrapper class. Caché Activate transparently creates an instance of the appropriate ActiveX component within the same process. When you invoke the methods of the wrapper class, it automatically dispatches them to a method of the appropriate ActiveX interface.

For details, see the manual *Using the Caché ActiveX Gateway*.

7

Development Tools and Database Utilities

Caché includes a number of application development tools as well as database administration utilities. On a Windows system, these are accessible from the Caché cube icon within the Windows task bar. (Right-click on the icon. If you do not see this icon, open the Windows **Start** menu, find the Caché folder under Programs, and click on the Caché entry.)

The various graphical utilities are client/server applications whose clients run on Windows systems but can be used with remote Windows, Linux, UNIX, and OpenVMS server systems.

The Caché development tools and utilities include:

Caché Studio

Caché Studio is the primary development environment for Caché. The Studio lets you create class definitions, Caché Server Pages, and routines using a full-featured editor. Studio includes sophisticated syntax checking and coloring, graphical debugging, and a point-and-click class inspector. Caché Studio works directly with the Caché database (it is a Caché application) and offers multideveloper support as well as source control system integration. You can customize Studio through the use of Studio templates. For more information, see *Using Caché Studio.*

Caché System Management Portal

The **System Management Portal** provides a Web-based interface for managing a Caché site. The management portal includes tools for system administrators, security managers, database operators, and any others needing access to Caché. Its administrative features allow you to set up a Caché system, view and alter its configuration parameters, adjust system settings, and create and edit databases and namespaces. Its database-browsing features include the ability to browse the contents of a Caché database, to examine data (globals) in the data engine; and to browse classes and routines — as well as monitoring database activity and performing backup operations. Its security-related features allow you to add, alter, and remove users, roles, resources, privileges, and to perform other security management tasks. Its SQL-related features provide a graphical, SQL-based view of a Caché database; you can use it to manage SQL roles and permissions, browse table and view definitions, execute ad hoc SQL queries, manage the query cache, and import and export data. For more information, see *Caché System Administration Guide.*

Caché Terminal

Caché Terminal provides an interactive, command-line interface to Caché. You can use Terminal for direct interactions with the Caché database engine. The Terminal is an important tool for testing and troubleshooting applications. For information, see the manual *Using Caché Terminal*.

Caché DocBook

The **Caché DocBook application** provides a fully searchable, Web-based interface to the documentation for Caché or any other installed InterSystems product. For an introduction to this application, see the "Basic Features" chapter of *Using InterSystems Documentation*.

Index