# Table of Contents

<table>
<thead>
<tr>
<th>Chapter One</th>
<th>Data Modeling – Relational or Object Access</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relational Technology</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Object Technology and Object Databases</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Object vs. Relational Access</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Overview of the Caché Object Data Model and Object Programming</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Key Object Concepts</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Why Choose Objects for Your Data Model?</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Object Data Storage – Plus Relational Access</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Not Only SQL</td>
<td>9</td>
</tr>
<tr>
<td>Chapter Two</td>
<td>Caché’s Multidimensional Data Server</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Integrated Database Access</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Multidimensional Data Model</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>SQL Access</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Caché Objects</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Transactional Bit-Map Indexing</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Word-Aware Text Searching</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>InterSystems iKnow Technology</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Enterprise Cache Protocol for Distributed Systems</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Disaster Recovery and High Availability</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Security Model</td>
<td>27</td>
</tr>
<tr>
<td>Chapter Three</td>
<td>Caché’s Application Server</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>The Caché Virtual Machine and Scripting Languages</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Caché ObjectScript</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Basic</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>MV Basic</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>C++</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Java</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Caché eXTreme for Java</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Caché and .NET</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Caché and XML</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Caché and Web Services</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Caché and MultiValue</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Migrating Applications to Caché</td>
<td>47</td>
</tr>
<tr>
<td>Chapter Four</td>
<td>Building Rich Web Applications Fast with InterSystems Zen Technology</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>CSP Technology</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Class Architecture of Web Pages</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Zen and Component-Based Web Pages</td>
<td>50</td>
</tr>
</tbody>
</table>
The computing world has entered the "post-relational" era

Thirty years ago, relational databases were hailed as a great innovation. Instead of monolithic legacy databases, each with its unique data schema, data would be stored in a tabular format, and be accessible to anyone who knew SQL. Relational databases were highly successful, and SQL became a common standard for database access. However, as is common with older technologies, relational databases have limitations that reduce their applicability to today's world – primarily in the realms of performance/scalability, ease of use, and fit with today's development technologies.

The usage and complexity of computer applications are exploding, and today's systems increasingly have processing requirements that outstrip the capabilities of relational technology. Many key applications that demanded high performance and scalability never made the transition to relational databases, and today even simple applications are beginning to approach the limits of traditional relational technology.

"Impedance mismatch" between relational databases and today's development technologies has become a serious problem – making development more complex and the chances of failure greater. While the simplicity of tabular structures supports an elegant query language (SQL), it is difficult to decompose real-world data structures into such simplistic rows and columns. The result is a huge number of tables whose relationships are hard to remember and hard to use – rows and columns are simple, but the pervasive need to program left outerjoins, stored procedures, and triggers is not.

Modern applications are usually written using object technology, which enables a faster and more intuitive way of describing and using information. Development is faster, and reliability increases. Unfortunately, objects are not natively compatible with relational databases. The advantages of object technology get blunted when the resulting database objects have to be forced into the two-dimensional relational model.

Today's transaction processing applications have requirements that outstrip the capabilities of relational technology – they must span large networks, service thousands of clients, but still provide superb performance, Web compatibility, and simple operations at low cost. And they must be developed quickly!
Advanced database technology for breakthrough applications.

Caché is a new generation of ultra-high-performance database technology. It combines an object database, high-performance SQL, and powerful multidimensional data access -- all of which can simultaneously access the same data. Data is only described once in a single integrated data dictionary and is instantly available using all access methods. Caché provides levels of performance, scalability, rapid programming, and ease of use unattainable by relational technology.

But Caché is much more than a pure database technology. Caché includes an Application Server with advanced object programming capabilities, the ability to easily integrate with a wide variety of technologies, and an extremely high-performance runtime environment with unique data caching technology.

Caché comes with several built-in scripting languages: Caché ObjectScript, a powerful yet easy-to-learn object-oriented programming language; Caché Basic, a superset of the widespread Basic programming language including extensions for powerful data access and object technology; and Caché MVBasic, a variant of Basic used by MultiValue applications (sometimes referred to as Pick applications). Other languages, such as Java, C#, and C++, are supported through direct call-in and other interfaces, including ODBC, JDBC, .NET, and a Caché-provided object interface that allows accessing the Caché database and other Caché facilities as properties and methods.

Caché also goes beyond traditional databases by incorporating a rich environment for developing sophisticated browser-based (Web) applications. InterSystems Zen technology allows the rapid development and execution of rich, highly interactive Web pages. Thousands of simultaneous Web users can access database applications, even on low-cost hardware.

For non-browser-based applications, the user interface is typically programmed in one of the popular client user interface technologies, such as Java, .NET, C#, or C++. Best results (fastest programming, greatest performance, and lowest maintenance) are usually obtained by performing all of the rest of the development within Caché. However, Caché also provides extremely high levels of interoperability with other technologies and supports all of the most commonly used development tools, so a wide range of development methodologies are available.
Data Modeling – Relational or Object Access

Early in the process of designing a new application, developers must decide upon their approach towards data modeling. For most, this comes down to a choice between the traditional modeling of data as relational tables and the newer approach of modeling as objects. Faced with the need to handle complex data, many developers believe that modeling with objects is a more effective approach.

Of course, when moving an existing application to Caché, the first step is to migrate the existing data model. There are easy ways to import data models from various relational or object representations so that the result is a standard Caché data definition. Once migrated to Caché, data can be simultaneously accessed as objects, relational tables, and multidimensional arrays.

Caché supports both SQL and object data access, and at times each is appropriate. To understand the uses of each and why data modeling with objects is generally preferred by modern-day developers, it is useful to understand how and why each has developed.
Relational Technology

In the early days of computing, information processing was done on huge mainframe systems and data access was, for the most part, limited to IT professionals. Databases tended to be homegrown, and retrieving data effectively required a thorough knowledge of the database. If users wanted a special report, they usually had to ask an overworked central staff to write it, and it usually wasn't available in time to influence decisions.

Although relational technology was originally developed in the 1970s on the mainframe, it remained largely a research project until it began to appear in the 1980s on mini-computers. With the advent of PCs, the world entered a more "user-centric" era of computing with more user-friendly report writers based on SQL – the query language introduced by relational technology. Users could now produce their own reports and ad hoc queries of the database, and relational usage exploded.

SQL allows a consistent language to be used to ask questions of a wide variety of data. SQL works by viewing all data in a very simple and standardized format - a two-dimensional table with rows and columns. While this simple data model allowed the construction of an elegant query language with which to ask questions, it came with a severe price. The inherent complexity of real-world data relationships doesn't fit naturally into simple rows and columns, so data is often fragmented into multiple tables that must be "joined" in order to complete even simple tasks. This results in two problems: a) queries can become very difficult to write due to the need to "join" many tables (often with complex outerjoins); and b) the processing overhead required when relational databases have to deal with complex data can be enormous.

SQL has become a standard for database interoperability and reporting tools. However, it is important to understand that while SQL grew out of relational databases, it need not be constrained by them. Caché supports standard SQL as a query and update language, using a much stronger multidimensional database technology, and it is extended to include object capabilities.
Object Technology and Object Databases

Object programming and object databases are a practical result of work to simulate complex activities of the brain. It was observed that the brain is able to store very complex and different types of data and yet still manipulate such seemingly different information in common ways. To support that simulation, very complex behavior needed to be implemented in programs while hiding that complexity – supporting simpler, more generalized and understandable logic with adaptable, reusable functionality. Clearly, these characteristics are also true of today’s leading-edge applications, and a technology that lets developers work in a natural manner that is more similar to how humans think is a huge advantage.

Object vs. Relational Access

In object technology, the complexity of the data is contained within the object, and the data is accessed by a simple consistent interface. In contrast, relational technology also provides a simple consistent interface, but because it does nothing to manage real-world data complexity – the data is scattered among multiple tables – the user or programmer is responsible for constantly dealing with that complexity.

Because objects can model complex data simply, object programming is the best choice for programming complex applications. Similarly, object access of the database is the best choice for inserting and updating the database (i.e., for transaction processing).

Caché complements object access with an object-extended SQL query language. SQL is a powerful language for searching a database and is widely used by reporting tools. However, we believe SQL is best suited for that purpose – queries and reports – rather than for transaction processing (for which it is cumbersome and often inefficient). Caché SQL’s object extensions eliminate much of the cumbersome join syntax, making SQL even easier to use.

Overview of the Caché Object Data Model and Object Programming

The Caché object model is based on the ODMG (Object Database Management Group) standard and supports many advanced features, including multiple inheritance.

Object technology attempts to mirror the way that humans actually think about and use information. Unlike relational tables, objects bundle together both data and code. For example, an Invoice object might have data, such as an invoice number and a total amount, and code, such as Print().
Conceptually, an object is a package that includes that object's data values ("properties") and a copy of all of its code ("methods"). An object's methods send messages to communicate with other objects. To reduce storage, it is common for objects of the same class to share the same copy of code (e.g., it would be unrealistic for each Invoice object to have its own private copy of code). Also, in Caché, method calls typically result in efficient function calls rather than enduring the overhead of passing messages. However, these implementation techniques are hidden from the programmer; it is always accurate to think in terms of objects passing messages.

What is the difference between an object and a class? A class is the definitional structure and code provided by the programmer. It includes a description of the nature of the data (its "type") and how it is stored as well as all of the code, but it does not contain any data. An object is a particular "instance" of a class. For example, invoice #123456 is an object of the Invoice class.
Key Object Concepts

**Inheritance** is the ability to derive one class of objects from another. The new class (a subclass) contains all of the properties and methods of its superclass, as well as additional properties and methods unique to it. Objects of the subclass can be thought of as having an "is a" relationship to its superclass. For example, a dog "is a" mammal, so it makes sense for the Dog class to inherit all the properties and methods of the Mammal class plus have additional properties and methods such as a DogTagNumber. A subclass may also override an inherited definition (e.g., the Print() method for a subclass of the Invoice class may be different from the Print() method of Invoice). Inheritance promotes reusability of code and makes it easier to introduce major improvements.

**Multiple Inheritance** means a subclass can be derived from more than one superclass. For example, a dog "is a" mammal and "is a" pet, so the object class "Dog" can inherit the properties and methods of both the "Mammal" class and the "Pet" class.

**Encapsulation** means that objects can be viewed as a sort of "black box". Public properties and methods can be accessed by methods of any class, whereas private properties and methods can only be accessed by methods of the same class. Thus, the application doesn't need to know the internal workings of an object – it deals only with the public properties and methods. The power of encapsulation is that programmers can improve the inner workings of a class without affecting the rest of the application.

**Polymorphism** refers to the fact that methods used in multiple classes can share a common interface, even if the underlying implementation is different. For example, suppose the classes Letter, Mailing Label, and ID Badge all contain a method called Print( ). To print, an application doesn't need to know which type of object it is accessing – it merely calls the object’s Print( ) method.

Why Choose Objects for Your Data Model?

For new database applications, most developers choose to use object technology because they can develop complex applications more rapidly and more easily modify them later. Object technology provides many benefits:

- Objects support a richer data structure that more naturally describes real-world data.
- Programming is simpler – it is easier to keep track of what you are doing and what you are manipulating.

The Caché Advantage

Caché is fully object-enabled, providing all the power of object technology to developers of high-performance transaction processing applications.

**Intuitive Data Modeling:**
Object technology lets developers think about and use information – even extremely complex information – in simple and realistic ways, thus speeding the application development process.

**Rapid Application Development:**
The object concepts of encapsulation, inheritance, and polymorphism allow classes to be reused, re-purposed, and shared between applications, enabling developers to leverage their work over many projects.

Polymorphism refers to the fact that methods used in multiple classes can share a common interface, even if the underlying implementation is different. For example, suppose the classes Letter, Mailing Label, and ID Badge all contain a method called Print( ). To print, an application doesn't need to know which type of object it is accessing – it merely calls the object’s Print( ) method.
Customized versions of classes can easily replace standard ones, making it easier to customize an application.

The black box approach of encapsulation means programmers can improve the internal workings of objects without affecting the rest of the application.

Objects provide a simple way to connect different technologies and different applications.

Object technology is a natural match with graphical user interfaces.

Many new tools assume object technology.

Objects provide a good insulation between the user interface and the rest of the application. Thus, when it becomes necessary to adopt a new user interface technology (perhaps some currently unforeseen future technology), you can reuse most of your code.

**Object Data Storage ...**

Unfortunately, although many applications are now being written with object programming languages, they often try to force object data into flat relational tables. This significantly impairs the advantages of object technology.

Caché provides a multidimensional data structure that naturally stores rich object data. The result is faster data access and faster programming.

**... Plus Relational Access**

Of course, many tools (such as report writers) use SQL, not object technology, for accessing data.

A unique feature of Caché is that whenever a database object class is defined, Caché automatically provides full SQL access to that data. Thus, with no additional work, SQL-based tools will immediately work with Caché data, and even they will experience the high-performance advantage of the Caché Multidimensional Data Server.

The reverse is also true. When a DDL definition of a relational database is imported, Caché automatically generates an object description of the data, enabling immediate access as objects, as well as through SQL.

The Caché Unified Data Architecture keeps these access paths synchronized; there is only one data description to edit.

**Not Only SQL**

Caché also allows direct access to its multidimensional data structures. This enables Caché to be used as a NoSQL or "Not Only SQL" database in situations where that is desirable.
Caché’s high-performance database uses a multidimensional data engine that allows efficient and compact storage of data in a rich data structure. Objects and SQL are implemented by specifying a unified data dictionary that defines the classes and tables and provides a mapping to the multidimensional structures – a mapping that can be automatically generated. Caché also allows direct multidimensional data access.

Integrated Database Access

Caché gives programmers the freedom to store and access data through objects, SQL, or direct access to multidimensional structures. Regardless of the access method, all data in Caché’s database is stored in Caché’s multidimensional arrays.

Once the data is stored, all three access methods can be simultaneously used on the same data with full concurrency.

A unique feature of Caché is its Unified Data Architecture. Whenever a database object class is defined, Caché automatically generates an SQL-ready relational description of that data. Similarly, if a DDL description of a relational database is imported into the Data Dictionary, Caché automatically generates both a relational and an object description of the data, enabling immediate access as objects. Caché keeps these descriptions coordinated; there is only one data definition to edit. The programmer can edit and view the dictionary both from an object and a relational table perspective.

Caché automatically creates a mapping for how the objects and tables are stored in the multidimensional structures, or the programmer can explicitly control the mapping.

The Caché Advantage

**Flexibility:** Caché’s data access modes – Object, SQL, and multidimensional – can be used concurrently on the same data. This flexibility gives programmers the freedom to think about data in the way that makes most sense and to use the access method that best fits each program’s needs.

**Less Work:** Caché’s Unified Data Architecture automatically describes data as both objects and tables with a single definition. There is no need to code transformations, so applications can be developed and maintained more easily.

**Leverage Existing Skills and Applications:** Programmers can leverage existing relational skills and introduce object capabilities gradually into existing applications as they evolve.
Multidimensional Data Model

At its core, the Caché database is powered by an extremely efficient multidimensional data engine. The built-in Caché scripting languages support direct access to the multidimensional structures – providing the highest performance and greatest range of storage possibilities – and many applications are implemented entirely using this data engine directly. Direct "global access" is particularly common when there are unusual or very specialized structures and no need to provide object or SQL access to them, or where the highest possible performance is required.

There is no data dictionary, and thus no data definitions, for the multidimensional data engine.

Rich Multidimensional Data Structure

Caché’s multidimensional arrays are called "globals". Data can be stored in a global with any number of subscripts. What's more, subscripts are typeless and can hold any sort of data. One subscript might be an integer, such as 34, while another could be a meaningful name, like "LineItems" – even at the same subscript level.

For example, a stock inventory application that provides information about item, size, color, and pattern might have a structure like this:

```
^Stock(item,size,color,pattern) = quantity
```

Here's some sample data:

```
^Stock("slip dress",4,"blue","floral") = 3
```

With this structure, it is very easy to determine if there are any size 4 blue slip dresses with a floral pattern – simply by accessing that data node. If a customer wants a size 4 slip dress and is uncertain about color and pattern, it is easy to display a list of all of those by cycling through all of the data nodes below:

```
^Stock("slip dress",4).
```

In this example, all of the data nodes were of a similar nature (they stored a quantity), and they were all stored at the same subscript level (four subscripts) with similar subscripts (the third subscript was always text representing a color). However, they don’t have to be. Data nodes may have a different number or type of subscripts, and they may contain different types of data.

Here's an example of a more complex global with invoice data that has different types of data stored at different subscript levels:

```
^Invoice(invoice #,"Customer") = Customer information
^Invoice(invoice #,"Date") = Invoice date
^Invoice(invoice #,"Items") = # of Items in the invoice
^Invoice(invoice #,"Items",1,"Part") = part number of 1st Item
^Invoice(invoice #,"Items",1,"Quantity") = quantity of 1st Item
^Invoice(invoice #,"Items",1,"Price") = price of 1st Item
^Invoice(invoice #,"Items",2,"Part") = part number of 2nd Item
etc.
```
Multiple Data Elements Per Node

Often only a single data element is stored in a data node, such as a date or quantity, but sometimes it is useful to store multiple data elements together in a single data node. This is particularly useful when there is a set of related data that is often accessed together. It can also improve performance by requiring fewer accesses of the database.

For example, in the above invoice, each item included a part number, quantity, and price all stored as separate nodes, but they could be stored as a list of elements in a single node:

```
^Invoice(invoice #,"LineItems",item #).
```

To make this simple, Caché supports a function called $list(), which can assemble multiple data elements into a length delimited byte string and later disassemble them. Elements can in turn contain sub-elements, etc.

Logical Locking Promotes High Concurrency

In systems with thousands of users, reducing conflicts between competing processes is critical to providing high throughput. One of the biggest conflicts is between transactions wishing to access the same data.

Caché processes don't lock entire pages of data while performing updates. Instead, because transactions require frequent access or changes to small quantities of data, database locking in Caché is done at a logical level. Database conflicts are further reduced by using atomic addition and subtraction operations, which don't require locking. (These operations are particularly useful in incrementing counters used to allocate ID numbers and for modifying statistics counters.)

With Caché, individual transactions run faster, and more transactions can run concurrently.

Variable Length Data in Sparse Arrays

Because Caché data is 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.

Declarations and Definitions Aren't Required

Caché multidimensional arrays are inherently typeless, both in their data and subscripts. No declarations, definitions, or allocations of storage are required. Global data simply pops into existence as data is inserted.

Namespaces

In Caché, data and code are stored in disk files with the name CACHE.DAT (only one per directory). Each such file contains numerous "globals" (multidimensional arrays). Within a file, each global name must be unique, but different files may contain the same global name. These files may be loosely thought of as databases.
Rather than specifying which database file to use, each Caché process uses a "namespace" to access data. A namespace is a logical map that maps the names of multidimensional global arrays and code to databases. If a database is moved from one disk drive or computer to another, only the namespace map needs to be updated. The application itself is unchanged.

Usually, other than some system information, all data for a namespace is stored in a single database. However, namespaces provide a flexible structure that allows arbitrary mapping, and it is not unusual for a namespace to map the contents of hundreds of databases, including some on other computers.

---

**The Caché Advantage**

**Performance:** By using an efficient multidimensional data model with sparse storage techniques instead of a cumbersome maze of two-dimensional tables, data access and updates are accomplished with less disk I/O. Reduced I/O means that applications will run faster.

**Scalability:** The transactional multidimensional data model allows Caché-based applications to be scaled to many thousands of clients without sacrificing high performance. That's because data access in a multidimensional model is not significantly affected by the size or complexity of the database in comparison to relational models. Transactions can access the data they need without performing complicated joins or bouncing from table to table.

Caché’s use of logical locking for updates instead of locking physical pages is another important contributor to concurrency, as is its sophisticated data caching across networks.

**Rapid Development:** With Caché, development occurs much faster because the data structure provides natural, easily understood storage of complex data and doesn’t require extensive or complicated declarations and definitions. Direct access to globals is very simple, allowing the same language syntax as accessing local arrays.

**Cost-Effectiveness:** Compared to similarly sized relational applications, Caché-based applications require significantly less hardware and no database administrators. System management and operations are simple.
SQL Access

SQL is the query language for Caché, and it is supported by a full set of relational database capabilities – including DDL, transactions, referential integrity, triggers, stored procedures, and more. Caché supports access through ODBC and JDBC (using a pure Java-based driver). SQL commands and queries can also be embedded in Caché ObjectScript and within object methods.

SQL accesses data viewed as tables with rows and columns. Because Caché data is actually stored in efficient multidimensional structures, applications that use SQL achieve better performance with Caché than with traditional relational databases.

Caché supports, in addition to the standard SQL syntax, many of the commonly used extensions in other databases so that many SQL-based applications can run on Caché without change – especially those written with database independent tools. However, vendor-specific stored procedures will require some work, and InterSystems has translators to help with that work.

Caché SQL includes object enhancements that make SQL code simpler and more intuitive to read and write.

TRADITIONAL SQL

```
SELECT
    SC.FullName, SM.Descr, MS.Value,
    SI.InvDate, SI.InvNumber
FROM
    MainSales MS, SalesItem SI, SalesProduct SP, SalesCustomer SC, SalesMarket SM
WHERE
    SI.SalesItemID *= MS.SalesItem
    AND SP.SalesProductID *= MS.Product
    AND SC.SalesCustomerID *= MS.Customer
    AND SM.SalesMarketID *= SC.SalesMarket
    AND SP.Descr = 'Hammer'
```

OBJECT EXTENDED SQL

```
SELECT
    Customer->FullName,
    Customer->SalesMarket->Descr, Value,
    SalesItem->InvDate, SalesItem->InvNumber
FROM
    MainSales
WHERE
    Product->Descr = 'Hammer'
```
Accessing Relational Databases with the Caché Relational Gateway

The Caché Relational Gateway enables an SQL request that originates in Caché to be sent to other (relational) databases for processing. Using the Gateway, a Caché application can retrieve and update data stored in most relational databases.

Optionally, the Gateway allows Caché database classes to transparently use relational databases. However, applications will run faster and be more scalable if they access Caché's post-relational database.

The Caché Advantage

**Faster SQL**: Relational applications can enjoy significantly enhanced performance by using Caché SQL to tap into Caché's efficient post-relational database.

**Faster Development**: In Caché, SQL queries can be written more intuitively, using fewer lines of code.

**Compatibility with Relational Applications and Report Writers**: Caché's native ODBC and JDBC drivers provide high-performance access to the Caché database for relational applications and reporting tools. The Caché Relational Gateway enables Caché applications to use SQL to access other (relational) databases.
Caché Objects

Caché's object model is based upon the ODMG standard. Caché supports a full array of object programming concepts, including encapsulation, embedded objects, multiple inheritance, polymorphism, and collections.

The built-in Caché scripting languages directly manipulate these objects, and Caché also exposes Caché classes as Java, .NET, and C++ classes. Caché classes can also be automatically enabled for XML and SOAP support by simply clicking a button in the Studio IDE. As a result, Caché objects are readily available to every commonly used object technology.

There are several ways for a program outside of the Caché Application Server to access Caché classes:

1. Any Caché class can be projected as a class in the native language. When a Java, C++, C#, or other program accesses a Caché object, it calls a template of the class in the native language. That template class (which is automatically generated by Caché) communicates with the Caché Application Server to invoke methods on the Caché server and to access or modify properties. State for the Caché objects is maintained in the Caché Application Server. To speed execution and reduce messaging, Caché caches a copy of the object's data on the client and piggybacks updates with other messages when possible.

2. Caché eXTreme technology can be used for database classes in which the native language template class directly accesses the database – bypassing the Application Server. The object's state is not kept on the Application Server; the in-memory properties are only maintained in the client. This approach provides significantly higher throughput but less functionality, since server-side instance methods of the class (i.e., methods that need access to the in-memory properties) cannot be invoked. Caché eXTreme is available for C++ and Java.

3. InterSystems' Jalapeño technology allows Java developers to first create Java database classes just like any other POJO (plain old Java object) class in their IDE of choice and then have Caché automatically generate a database schema and corresponding Caché class. Using this approach, the Java class is unchanged, and the application continues to access its properties and methods. Caché provides a library class ("ObjectManager") with an API that is used to store and retrieve database objects and issue queries.

With each of these three approaches, the object appears to be local to the user program. Caché transparently handles all communications, using either in-process or via TCP.
Method Generators

Caché includes a number of unique advanced object technologies – one of which is method generators. A method generator is a method that executes at compile time, generating code that can run when the program is executed. A method generator has access to class definitions, including property and method definitions and parameters, which allow it to generate a method that is customized for the class. Method generators are particularly powerful in combination with multiple inheritance – functionality can be defined in a multiply inherited class that customizes itself to the subclass.

The Caché Advantage

Caché is fully object-enabled, providing all the power of object technology to developers of high-performance transaction processing applications.

Rapid Application Development:
Object technology is a powerful tool for increasing programmer productivity. Developers can think about and use objects – even extremely complex objects – in simple and realistic ways, thus speeding the application development process. Also, the innate modularity and interoperability of objects simplify application maintenance, and lets programmers leverage their work over many projects.

Natural Development:
Database objects appear as objects native to the language being used by the developer. There is no need to write tedious code to decompose objects into rows and columns and later re-assemble them.
Transactional Bit-Map Indexing

Caché uniquely provides Transactional Bit-Map Indexing, which can radically increase performance of complex queries giving fast data warehouse query performance on live data.

Database performance is critically dependent on having indexes on properties that are frequently used in searching the database. Most databases use indexes that, for each possible value of the column or property, maintain a list of the IDs for the rows/objects that have that value.

A bit-map index is another type of index. Bit-map indexes contain a separate bit-map for each possible value of a column/property, with one bit for each row/object that is stored. A 1 bit means that the row/object has that value for the column/property.

The advantage of bit-map indexes is that complex queries can be processed by performing Boolean operations (AND, OR) on the indexes – efficiently determining exactly which instances (rows) fit the query conditions, without searching through the entire database. Bit-map indexes can often boost response times for queries that search large volumes of data by a factor of 100 or more.

Bit-maps traditionally suffer from two problems: a) they can be painfully slow to update in relational databases, and b) they can take up far too much storage. Thus, with relational databases, they are rarely used for transaction processing applications.

With Transactional Bit-Map Indexing, Caché leverages its multidimensional data structures to eliminate these two problems. Updating Caché’s bit-maps is often faster than traditional indexes, and they utilize sophisticated compression techniques to radically reduce storage. Caché also supports sophisticated “bit-slicing” techniques. The result is ultra-fast bit-maps that can often be used to search millions of records in a fraction of a second on an online transaction-processing database. Business intelligence and data warehousing applications can work with "live" data.

Caché offers both traditional and transactional bit-map indexes. Caché also supports multi-column indexes. For example, an index on State and CarModel can quickly identify everyone who has a car of a particular type that is registered in a particular state.
Word-Aware Text Searching

Through Word-aware indexing, Caché supports free text searching in which queries can search for text containing words of interest, even though the actual words in the text may be variants of the search words. Word-Aware algorithms are specific to the natural language being used. Word-Aware searching is available for a wide range of natural languages, including English, French, German, Italian, Japanese, Portuguese, and Spanish. Others are being added.

<table>
<thead>
<tr>
<th>WHERE Description</th>
<th>Caché finds “search”, “searched”, “searching”, …</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Contains (‘search’)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WHERE Description</th>
<th>Caché finds “close”, “closed”, … But not “closet” or “disclose”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Contains (‘close’)</td>
<td></td>
</tr>
</tbody>
</table>

The Caché Advantage

**Powerful Text Searches:**
Unstructured text, such as physicians’ notes or documents, can be easily searched for keywords and related words.

**Extremely Rapid Searches:**
Coupling Word-aware indexing with Caché bit-map technology, searching of massive quantities of text can be performed in a fraction of a second.
InterSystems iKnow Technology

InterSystems iKnow technology allows you to analyze and index text and other unstructured data types, identifying important knowledge, concepts, and relationships. Unlike most other semantic and search technologies, iKnow will automatically indicate the most interesting meaningful elements in the data without needing any input from the user, not even a search term.

- **Smart Indexing**
  Smart indexing analyzes and transforms unstructured text into an understandable network of relationships and concepts without any need for pre-defined dictionaries, taxonomies, or ontologies. Smart indexing provides insight into what's relevant, what's related, and what's representative within large volumes of unstructured text, without needing input of a search term.

  Smart indexing works with a number of different languages. It can also identify concepts (recurring patterns) within unstructured data that is not traditional text.

- **Smart Matching**
  Smart matching links the results of smart indexing to existing knowledge specific to a domain, organization, or industry. Matching is based on meaningful concepts and their combinations, not just words, and includes exact, partial, and “scattered” matches.

- **Smart Interpretation**
  Smart interpretation applies analytics and/or business rules to the results derived from smart indexing and smart matching. (May require use of InterSystems Ensemble- or InterSystems DeepSee.)
The Caché Advantage

**Built-in Capabilities for Analyzing and Indexing Unstructured Data:**
InterSystems iKnow technology is built into Caché. You can access all your data – structured and unstructured – without resorting to third-party solutions or tools.

**Minimal Up-front Work:**
Unlike other semantic search tools, InterSystems iKnow technology can find concepts and relationships without having them pre-defined. Users don’t necessarily need to know what they are looking for before they look for it.

**Multilingual:**
InterSystems iKnow technology works with multiple languages, even if languages are mixed within a document.

**Multipurpose:**
InterSystems iKnow technology can be used to solve different kinds of problems: Identifying the most interesting elements in a huge pile of documents, automatically routing information based on its similarity to other information, summarizing large amounts of text into comprehensive summaries, etc.
Enterprise Cache Protocol for Distributed Systems

Scalable Performance in Distributed Systems

InterSystems’ Enterprise Cache Protocol (ECP) is an extremely high-performance and scalable technology that enables computers in a distributed system to use one another’s databases. The use of ECP requires no application changes – applications simply treat the database as if it were local.

Here’s how ECP works: Each Caché Application Server includes its own Caché Data Server, which can operate on data that resides in its own disk systems or on blocks that were transferred to it from another Caché Data Server by ECP. When a client makes a request for information that is maintained on a remote Data Server, the Application Server will attempt to satisfy the request from its local cache. If it cannot, it will request the necessary data from the remote Data Server. The reply includes the database block(s) where that data was stored. These blocks are cached on the Application Server, where they are available to all applications running on that server. ECP automatically takes care of managing cache consistency across the network and propagating changes back to Data Servers.

The performance and scalability benefits of ECP are dramatic. Clients enjoy fast responses because they frequently use locally cached data. And caching greatly reduces network traffic between the database and application servers.

Easy to Use – No Application Changes

The use of ECP is transparent to applications. Applications written to run on a single server run in a multi-server environment without change. To use ECP, the system manager simply identifies one or more Data Servers to an Application Server and then uses Namespace Mapping to indicate that references to some or all global structures (or portions of global structures) refer to that remote Data Server.

Configuration Flexibility

Every Caché system can function both as an Application Server and as a Data Server for other systems. ECP supports any combination of Application Servers and Data Servers and any point-to-point topology of up to 255 systems.
The Caché Advantage

**Massive Scalability:** Caché’s Enterprise Cache Protocol allows the addition of application servers as usage grows, each of which uses the database as if it were a local database. If disk throughput becomes a bottleneck, more Data Servers can be added and the database becomes logically partitioned.

**Higher Availability:** Because users are spread across multiple computers, the failure of an Application Server affects a smaller population. Should a Data Server "crash" and be rebooted, or there is a temporary network outage, the Application Servers can continue processing with no observable effects other than a slight pause. Configuring Data Servers as a fail-over hardware cluster with backup data servers can significantly enhance availability.

**Lower Costs:** Large numbers of low cost computers can be combined into an extremely powerful system supporting massive processing.

**Transparent Usage:** Applications don’t need to be written specifically for ECP – Caché applications can automatically take advantage of ECP without change.
Disaster Recovery and High Availability

Even in the most rigorous environments unexpected events can occur – hardware failure, power loss, or something as severe as a flood or other natural disaster – yet hospitals, telecommunications, and other critical operations cannot afford to be “down”. To meet such exacting standards, Caché is designed to recover gracefully from outages and offers a variety of failover and other options to reduce or eliminate the impact on users.

Caché Write-Image Journaling and other integrity features ensure database integrity for most types of hardware failures – including power outages – allowing rapid recovery while minimizing the impact on users.

Caché also provides advanced high-availability configuration options to further reduce or eliminate user impact, including:

- Database Mirroring
- Distributed ECP
- Failover Clusters

Database Mirroring

Mirroring is a high-availability configuration that replicates data on a separate disk in real time, with automatic failover.

A database mirror is a logical grouping of two Caché systems. Upon startup, the mirror automatically designates one of these two physically independent systems as the primary system; the other one automatically becomes the backup system. Mirrored databases are synchronized from the primary to the backup failover member in real time through a TCP channel. The backup system returns acknowledgments about receipt of mirrored data over a dedicated channel. The acknowledgment indicates, among other things, how up to date the backup failover member is.
External clients (language bindings, ODBC/JDBC/SQL clients, direct-connect users, etc.) connect to the mirror through the Mirror Virtual IP (VIP), which is specified during mirroring configuration. The Mirror VIP is automatically bound to an interface on the primary system of the Mirror. The configuration of a Mirror VIP is optional; if not specified, all external clients must connect directly to the running primary, and must have knowledge of both the failover members and their current role within the Mirror.

If the primary system fails, the backup system automatically takes over. When the Mirror is configured so that clients are logged on through the VIP, they do not know which of the mirror members is serving them. The failover is entirely transparent to users.

Caché can also update multiple (usually geographically separate) ASYNC mirror members, allowing you to build redundancy into your system and to aid in disaster recovery. Failover to an ASYNC member is not automatic.

Also, multiple mirrors can update one ASYNC member, creating a central data store, enabling real-time business intelligence using enterprise-wide data.

Using a Database Mirror with ECP

When used with a Database Mirror, Enterprise Cache Protocol (ECP) application servers have built-in knowledge of the members of the mirror, including the current primary. The application servers, therefore, do not rely on the Mirror VIP, but rather connect directly to the elected primary system.

If the primary member of the Mirror fails, the ECP application servers will view this as a server restart condition. The servers will simply reestablish their connections to the new primary failover member and continue processing their in-progress workload. Users will, at most, experience only a slight pause.

Failover Clusters

Using failover clustered hardware, data servers share access to the same disks, but only one is actively running Caché at a time. If the active server fails, Caché is automatically started on another server that takes over the processing responsibilities. The users can then sign back on to the new server.

An ECP data server can be configured as a failover cluster. If the primary data server crashes, the backup data server takes over.
Security Model

Caché is certified for Common Criteria EAL 3. The Caché security model is designed to support application deployment in three ways:

Caché security is based on authentication, authorization, auditing, and database encryption. Caché provides these security capabilities while minimizing the burden on application performance.

Authentication

Authentication is how users prove to Caché that they are who they say they are. (A “user” is not necessarily a human being. It could, for example, be a measurement device generating data or an application running on another system that is connected to 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.

- **LDAP**: Caché supports authentication through the Lightweight Directory Access Protocol (LDAP). In this case, Caché contacts an LDAP server to authenticate users, relying on its database of users and their associated information to perform authentication. The LDAP server also controls all aspects of password management, password policies, etc.

- **Passwords**: Caché prompts the user for a password and compares a hash of the provided password against the hash value it has stored.

- **Delegated Authentication**: Delegated authentication provides as a means for creating customized authentication mechanisms. The application developer entirely controls the content of delegated authentication code. Caché includes a template for creating custom authentication code.

- **Operating-system-based**: OS-based authentication trusts that the OS has verified the identity of each user, and uses that same identification for Caché purposes.
You can also allow all users to connect to Caché without performing any authentication.

Caché provides built-in support for two-factor authentication, which requires users to verify their identity via something they know and something they have. For example, when a user provides a password (something they know) the applications may send a text message to their cell phone (something they have). The text message would include an additional security code that must be entered before access to the application is granted.

Caché supports single sign-on by enabling participation in an OpenAM configuration.

**Authorization**

Once a user is authenticated, the next security-related operation is to determine what that user is allowed to use, view, or alter. This determination and control of access is known as authorization. The assignation and management of privileges are normally performed through the Caché Management Portal.

**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, such as READ, WRITE, or USE. Together, a resource and an associated, assigned permission are known as a privilege. In addition to the system-defined resources, the security administrator can create application-specific resources and use the same mechanisms for granting and checking permissions.

**Roles**

For simplicity, users are usually assigned one or more “roles” (e.g., “LabTech”, or “Payroll”), and the Security Administrator then grants privileges for a particular resource to those roles rather than to individual users. The user inherits all of the privileges granted to the roles it is assigned.

**Application-Assigned Roles**

It is often useful for a user to temporarily gain additional privileges rather than have them permanently assigned. For example, rather than the security administrator granting a broad set of privileges to a user (such as the ability to access and modify the payroll database), the user can instead be given just the privilege to access the payroll application, and that application can then elevate the user’s privileges while that application is being used.

To accomplish this elevation, roles can be assigned to applications. When that application is accessed, the user temporarily acquires additional roles. The additional roles may simply be a list that everyone authorized to use the application acquires, or the additional roles may be more customized, based on the roles the user already has.
This feature is particularly useful for Web-based applications. Following authentication and a determination that the user is authorized to use the application, the user temporarily gains the additional roles assigned to that application for the duration of that page request.

The Security Administrator can also designate specific routines as capable of performing role elevation to gain the additional roles of specified applications, after passing user-specified security tests. This facility is tightly controlled, and is the mechanism by which non-Web applications perform role elevation.

**Auditing**

Many applications, especially those that must comply with government regulations like HIPAA or Sarbanes-Oxley, need to provide secure auditing. In Caché, all system and application events are recorded in an append-only log, which is compatible with any query or reporting tool that uses SQL.

**Encryption: Data-at-Rest**

Caché supports two forms of database encryption:

- The Security Administrator can designate one or more CACHE.DAT files (databases) to be encrypted on disk. Everything in those files is then encrypted, including any indexes.

- Developers can use system functions to encrypt/decrypt data, which then may be stored in the database or transmitted. This feature can be used to encrypt sensitive data to protect it from other users that have read access to the database but not the key.

By default, Caché encrypts data with an implementation of the Advanced Encryption Standard (AES), a symmetric algorithm that supports keys of 128, 192, or 256 bits. Encryption keys are stored in a protected memory location. Caché provides full capabilities for key management.

Journal files are also encrypted.

**Encryption: Data-in-Motion**

Caché supports the use of SSL (Secure Sockets Layer) and its successor, TLS (Transport Layer Security), to secure connections of several types:

- When acting as a server, Caché accepts connections and establishes the use of SSL. This includes connections from Caché shadow destinations to Caché shadow sources.

- When acting as a client, Caché is able to connect to servers that require the use of SSL.

In all cases, Caché uses what is called an SSL/TLS configuration, which specifies the various characteristics of a Caché instance as part of an SSL/TLS connection.

More information about securing data-in-motion can be found in the “Caché and Web Services” section.
Caché’s Application Server

The Caché Application Server offers advanced object programming capabilities, provides sophisticated data caching, and integrates easy access to a variety of technologies. The Caché Application Server makes it possible to develop sophisticated database applications rapidly, operate them with high performance, and support them easily.

More specifically, the Caché Application Server provides:

- The Caché Virtual Machine which runs three built-in scripting languages – Caché ObjectScript, Caché Basic, and Caché MVBasic.
- Access to Caché Multidimensional Data Servers on the same and other computers with transparent routing.
- Connectivity software with client-side caching to permit rapid access to Caché Objects from all commonly used technologies, including Java, .NET, C++, C#, COM, and Delphi. Caché automatically performs the networking between the client and Application Server.
- Compatibility with SOAP and XML.
- SQL access using ODBC and JDBC, including sophisticated caching at the client and application server for high performance.
- Access to relational databases.
- Caché Server Pages and Zen for high-performance, easy-to-program Web applications.
- Caché Studio, an IDE to rapidly develop and debug applications with Caché.
- Code for the Scripting Languages is stored in the database and can be changed online, with changes automatically propagating to all application servers.

The Caché Virtual Machine and Scripting Languages

The core of the Caché Application Server is the extremely fast Caché Virtual Machine, which supports Caché's scripting languages.

- Caché ObjectScript is a powerful and easy-to-learn object-oriented language with extremely flexible data structures.
- Caché Basic provides an easy way for Visual Basic programmers to start using Caché. Similar to VBScript, Caché Basic supports objects and is extended to have direct access to the Caché Multidimensional Arrays.
- Caché MVBasic is the variant of the Basic programming language used in MultiValue (Pick) applications. MVBasic has been extended to support objects and have direct access to the Caché Multidimensional Arrays.
Database access within the Caché Virtual Machine is highly optimized. Each user process in the Caché Virtual Machine has direct access to the multidimensional data structures by making calls to shared memory that access a shared database cache. All other technologies (Java, C++, ODBC, JDBC, etc.) connect through the Caché Virtual Machine to access the database.

**Complete Interoperability**

Since Caché ObjectScript, Basic, and MVBasic are all implemented on the same Caché Virtual Machine, they are completely interoperable:

- Any object method can be written in any language – the same class can use all three languages.

- Each language’s function calls can access code written in the other languages.

- They share variables, arrays, and objects.

**Faster Development/Flexible Deployment**

By writing as much code as possible in these scripting languages, programmers can develop applications faster, and those applications will run significantly faster with greater scalability. Plus, code written in Caché ObjectScript, Basic, or MVBasic requires no changes to switch hardware or operating systems. Caché automatically handles all differences in operating system and hardware.

---

**The Caché Advantage**

**Rapid Application Development:** Development of complex database applications with Caché ObjectScript is radically faster than any other major language – often 10 to 100 percent faster. Faster also means the project has a better chance of succeeding – with fewer programmers – and being able to adjust rapidly as application needs change.

**Shorter Learning Curve:** Basic is perhaps the world’s best-known computer language. Developers who know Visual Basic can instantly start writing code in Basic, and the Caché object model is easily learned.

**Faster and More Scalable:** The Caché Virtual Machine with its direct access to the database provides faster applications that can scale to tens of thousands of users using low-cost hardware.

**Flexibility:** Code that runs in the Caché Virtual Machine can run on other hardware and operating systems without change. Code is stored in the database and automatically propagated to Application Servers.
Caché ObjectScript

Caché ObjectScript is a powerful object-oriented programming language designed for rapid development of database applications. Here are some of the key characteristics of the language.

**Overall Structure**

Caché ObjectScript is command-oriented; hence it has syntax such as:

```
set x=a+b
do rotate(a,3)
if (x>3)
```

There is a set of built-in system functions that are particularly powerful for text manipulation. Their names all start with the single “$” character to distinguish them from variable and array names. For example:

```
$extract(string,from,to) // get a set of characters from a string
$length(string) // determine the length of a string
```

Expressions use left-to-right operator precedence, just like most hand-held calculators, except when parentheses alter the order of evaluation.

**Flexible Data Storage**

One of the most unique characteristics of Caché ObjectScript is its highly flexible and dynamic data storage. Data may be stored in:

- Object properties
- Variables
- Sparse, multidimensional arrays that permit any type of data for subscripts
- Database files ("globals"), which are sparse multidimensional arrays

With rare exceptions, any place in the language where a variable can be used, an array, object property, or global reference could also be used.

In most computer languages, datatypes are an extension of hardware storage concepts (integer, float, char, etc.). However, Caché ObjectScript has the philosophy that humans don’t think using such storage types, and that these "computer-centric" datatypes simply impede rapid application development. Requiring declarations and dimension statements introduces far more errors than they help prevent (such as a 2-byte integer overflow, or when a string overflows its allocation of memory and corrupts other storage). However, object typing, such as Person, Invoice, or Animal, is viewed as highly valuable and consistent with the way humans think.
Thus, in Caché ObjectScript, object properties are strongly typed, but the other three types of storage (variables, arrays, and global nodes) are fully polymorphic, typeless entities that need not be declared or defined. They simply pop into existence as they are used and mold themselves to the data needs of what they are storing and how they are being used in an expression. Even arrays do not need any specification of size, dimension, type of subscripts, or data. For example, a developer might create an array called Person by simply setting:

```objectscript
set Person("Smith","John")= "I'm a good person"
```

In this example, data was stored in a two-dimensional array using string data for subscripts. Other data nodes in this array might have a different number of dimensions and might intermix strings, integers, or other types of data for subscripts. For example, one might store data in:

```objectscript
abc(3)
abc(3,-45.6,"Yes")
abc("Count")
```

all in the same array.

**Direct Access to the Database**

A direct reference to the database (a “global reference”) is essentially a multidimensional array reference preceded by the carat character “^”. That character indicates this is a reference to data stored in the database rather than to temporary process private data. Each such database array is called a “global”.

As with multidimensional arrays and variables, no declarations or definitions or reservations of storage are required to access or store data in the database; global data simply pops into existence as data is stored. For example, to store information in the database one might write:

```objectscript
set ^Person("Smith","John")= "I'm a very good person"
```

and later might retrieve it by code such as:

```objectscript
set x=^Person("Smith","John")
```

The programmer has complete flexibility in how to structure these global data arrays. (See the section on the Multidimensional Data Model.)
Object References

Caché objects implement the ODMG data model, with powerful extensions.

In Caché ObjectScript, an "oref" is used to access an object. (An oref is typically a variable whose value specifies which in-memory object is being referenced.) The oref is followed by a dot and then by the name of a property or method. Object references can be used wherever an expression can be used. For example:

```
set name=person.Name // "person" is a variable whose value is an oref
                    // the person's name is put into the variable "name"
if (person.Age>x)    // see if the person's age is greater than "x"
set money=invoice.Total() // "Total()" is a method that calculates the sum of
                          // all of the invoice's line items
```

Methods can also be executed with a DO command when no return value is needed. For example:

```
do part.Increment()    // "Increment()" is a method whose return value,
                        // if any, is not of interest
```

The oref is not the same as a database object ID. The object ID is a value that is permanently associated with a database object; it is used to retrieve and store a database object. Once an object is in memory, it is assigned a reusable oref value that is then used to access the object’s data. The next time that same database object is brought into memory it will probably be assigned a different oref value.

HTML and SQL Access

HTML for Web applications and SQL can be embedded in Caché ObjectScript code.

Calling Code

In some object languages, all code has to be part of some method. Caché ObjectScript doesn’t have that restriction – code may be directly called or called through object syntax.

Code is often called using the DO command:

```
do rotate(a,3)
```

Code that returns a value can also be called as a function.

For example,

```
set x=a+$$insert(3,y)
```
calls the programmer-written procedure or subroutine "insert".

Code can also be invoked as an object method:

```
set money=invoice.Total() // Total() returns the invoice total amount due
part.Increment()         // "Increment()" is a method whose return value,
                        // if any, is not of interest
```

Both call by value and call by reference are supported for parameters.
Routines

Caché ObjectScript code is fundamentally organized into a set of “routines”. Each routine (typically up to 32KB in size) is atomic in the sense that it can be independently edited, stored, and compiled. Routines are linked dynamically at run time; there is no separate linking step for the programmer. Routine code is stored in the database; thus, routines can be dynamically paged across the network rather than having to be installed on each computer.

Within a routine, code is organized as a set of procedures and/or subroutines. (An object method is a procedure, but it is accessed by a different syntax.)

When calling code that is within the same routine, only the procedure or subroutine name is needed. Otherwise, the routine name must be appended to it.

```objectscript
   do transfer() // calls "transfer" in the same routine
   do total^invoice() // calls "total" in the routine "invoice"
```

A procedure or subroutine that has a return value of interest should be called using the “$$” function syntax.

```objectscript
   set x=$$total^invoice() // calls the same "total" procedure but uses the return value
```

Routines can be edited and compiled through the Caché Studio.
Object Methods

Class definitions and their method codes are stored in global data files, and the Class Compiler compiles each class into one or more routines. Each method is simply a procedure in a routine, although it can only be invoked by object syntax. For instance, if the Patient class defines an Admit method and the Pat variable identifies a specific Patient object, then we call the Admit method for that object with the following syntax:

```
do Pat.Admit()       // Call the admit method for Patient
set x = Pat.Admit()   // Calls the same method but uses the return value
```

Procedures and Public/Private Variables

A procedure is a block of code within a routine that is similar to a function in other languages. A procedure consists of a name, a formal parameter list, a list of public variables, and a block of code delimited by "{}". For example:

```
Admit(x,y)[name,recnum] { ...code goes here } 
```

In Caché ObjectScript, some variables are public (common) and others are private to a particular procedure. Every variable that is used within a procedure is considered private to that procedure unless it is listed in the public list. In the above example, "name" and "recnum" access the public variables by those names, whereas all other variables exist only for this invocation of this procedure. Variable names that start with a "%" character are always implicitly public.

Procedures cannot be nested, although a procedure can contain subroutines.

Subroutines

Routines may also contain subroutines, which are lighter weight than procedures. A subroutine may contain a parameter list and it may return a value, but it does not have a public list or formal block structure. Subroutines may be embedded within procedures or be at the same level as a procedure in a routine.

Subroutines permit the calling of code using the same public/private set of variables as the caller, and they can be called quicker. A subroutine embedded within a procedure uses the same variable scope as the procedure and may only be called from within that procedure. Variable references in a subroutine that is not part of a procedure are all references to public variables.
Chapter Three

Basic

Basic is perhaps the world’s best-known application programming language. In Caché, Basic has been extended to support direct access of the Data Server’s core data structures – multidimensional arrays – as well as other Caché Application Server features. It directly supports the Caché Object Model using Visual Basic syntax, and runs in the Caché Virtual Machine.

Basic can be used either as methods of classes or as Caché routines (see the Caché ObjectScript description of routines). Basic can call Caché ObjectScript, and vice versa, with both languages accessing the same variables, arrays, and objects in process memory.

Arrays have been extended to be far more powerful:

- The presence of a “^” character preceding the array name indicates a reference to a database multidimensional array – persistent arrays that are shared with other processes.
- Subscripts can be of any datatype – strings, integers, decimal numbers, etc.
- Data can be stored at multiple subscript levels in the same array – for example, data could be stored at A(“colors”) and A(“colors”,3).
- Arrays do not have to be declared and are always sparse – Caché only reserves space as nodes are inserted.
- A Traverse function allows identification of the next (or previous) subscript at a given subscript level.

Other extensions include:

- Transaction processing commands to Start, Commit, and Rollback a transaction.
- An atomic increment function that can be used on the database.
- Extensions that provide better integration with the Caché Application Server capabilities.
Object Access with Basic

In Caché, classes are organized into packages, and class names include the package name followed by a period. For example, Payroll.Person is a Person class of the Payroll package. The Basic New command is used to create an object:

```
    person = New Payroll.Person()     // creates a new Person object
```

Basic has been extended with an OpenID command to access an existing object:

```
    person = OpenID Payroll.Person(54) // opens the Person object with OID 54
```

Here are some examples of code that access the person's properties:

```
    person.Name = "Smith, John"       // sets the person's name
    person.Home.City                  // references the person's home city
    person.Employer.Name              // brings the person's employer object into
    // memory and accesses the employer's name
```

Database classes can be saved to disk with the Save method. For example:

```
    person.Save()
```

will save the person, creating an object ID if it is the first time the object was stored. If related objects (such as the Employer) were also modified, they are automatically saved as well.
**MVBasic**

MVBasic is another scripting language offered in Caché that is a variant of Basic. However, it is intended to execute applications written for MultiValue (Pick) systems and therefore supports additional characteristics, including capabilities to access and manipulate MultiValue files.

MVBasic can be used either as methods of classes or as Caché routines (see the Caché ObjectScript description of routines). MVBasic can call Caché ObjectScript or Basic, and vice versa, with all three languages accessing the same variables, arrays, and objects in process memory.

Caché MVBasic has the same extensions as Caché Basic, including object access. However, because of possible ambiguity, the two character sequence “->” is used instead of using a period separator “.” in object references.

**C++**

Every Caché class can be projected as a C++ class, with methods corresponding to each property and method of the Caché class. To C++ programs, these classes look just like any other local C++ classes, and Caché automatically handles all communications between the client and server. Properties of the class are cached on the client, and C++ method calls invoke corresponding server side methods, including methods to store an object in the database and later retrieve it.

Caché also provides Caché eXTreme technology for C++. This is a “light binding” that allows C++ objects to inserted directly into Caché’s multidimensional database, enabling extremely high performance.

**Java**

Java is a popular object-oriented programming language, but because of its “write once, deploy anywhere” flexibility, it does not specify how data is written to, or read from, a database. When data needs to be persisted, Java developers must choose and implement data access methods.

Caché provides several ways for Java applications to access the Caché database:

- JDBC provides high-performance SQL access using a completely Java-based (Type 4) driver.
- Any Caché class can be projected as a Java class so properties and methods can be accessed as Java objects.
- InterSystems Jalapeño technology creates Caché classes from Java classes. Caché automatically provides methods to store and retrieve objects from the database without touching the developer’s classes.
- Caché eXTreme for Java exposes Caché data via the JNI (Java Native Interface).
- Java Persistent API (JPA) is enabled through both Hibernate and EclipseLink implementations to provide standard Java EE persistence support.
Caché eXTreme for Java

Caché eXTreme for Java exposes Caché’s enterprise and high-performance features to Java via the JNI (Java Native Interface). It enables “in-process” communication between Java and Caché, thereby providing extremely low-latency data storage and retrieval.

Caché eXTreme for Java gives developers a Java API (Application Programming Interface) that provides in-process communication for:

- Storing and retrieving relational data via JDBC (JDBC over JNI)
- Storing and retrieving Java objects
- Storing and retrieving multidimensional data

The Globals API option provides Java developers with direct access to Caché’s efficient multidimensional data structures.

The Caché Advantage

Very High-Performance Persistence: Caché eXTreme for Java speeds up database I/O for Java applications. It will be especially beneficial when building or enhancing Java applications that deal with streaming data, very high volumes of data, or rapidly changing data.

Multiple Modes of Data Access: Caché eXTreme for Java provides speedy “in-process” communication between Java and Caché via JDBC, object access, or direct multidimensional access.

Short Learning Curve: Caché eXTreme for Java uses Java standards to provide database access, so Java developers can work mostly within a Java environment. Typically, a Java developer with a year of experience can be productive in just a few days when using Caché eXTreme.

Enabling persistence for Java-based CEP applications

Most CEP (Complex Event Processing) applications cannot use persistent databases because the latency introduced by storing and retrieving information to/from disk is too great. Instead, they typically use in-memory databases, and run the risk of losing data or transactions. But by using Caché eXTreme for Java, Java-based CEP applications can persist data without sacrificing performance.
Caché and .NET

Because of its open and flexible data access, Caché works seamlessly with .NET. There are many ways of connecting the two, including objects, SQL, XML, and SOAP. Developers can create applications with the technologies they prefer. All of them will benefit from Caché’s superior performance and scalability.

ADO.NET

ADO.NET is a new incarnation of ADO, optimized for use in the .NET framework. It is intended to make .NET applications “database independent”, and generally uses SQL to communicate with databases. Through its relational data access, Caché provides native support for ADO.NET. It also supports Microsoft’s ODBC.NET and the read-only SOAP connectivity that is built into ADO.NET.

Web Services

There are two ways of using Web services in .NET. One is to send XML documents over HTTP. The other is to use the SOAP protocol to simplify the exchange of XML documents. Because Caché can expose data both ways, it works seamlessly with .NET Web services.

Caché Managed Objects

Caché can automatically generate .NET assemblies (or C# source code) from Caché classes. A plug-in for Visual Studio lets developers who prefer that environment to easily access Caché objects.
Caché and XML

Just as HTML is an Internet-compatible mark-up language for displaying data in a browser, XML is a mark-up language for exchanging data between applications. The structure of XML data is hierarchical and multidimensional, making it a natural match for Caché’s multidimensional data engine.

Exporting XML

All that is required to make a Caché class compatible with XML is to have it inherit from the %XMLAdaptor class that is included in Caché. This provides all the methods needed to:

- Create either a DTD (Document Type Definition) or an XML Schema for the class. Caché will automatically generate DTDs and Schemas, but developers who wish to customize the XML formatting of a class may do so.
- Automatically format an object’s data as XML, according to the defined DTD or Schema.

Importing XML

Caché comes with other classes that provide methods allowing developers to:

- Import XML Schemas and automatically create corresponding Caché classes.
- Import the data in XML documents as instances (objects) of Caché classes, via a simple API.
- Parse and validate XML documents via a built-in XML (SAX) parser.
Caché and Web Services

Web services enable the sharing of application functionality over a network, either worldwide via the Internet, or within an organization or system. The body of a Web service is an XML document, which is contained in an “envelope” usually formatted according to the SOAP (Simple Object Access Protocol) standard. Web services have an interface described in WSDL (Web Service Definition Language).

Any existing Caché class can be marked as a Web service simply by making it inherit from the appropriate system-level classes that come with Caché. There is also a Web Services Wizard for generating new Web services with just a few clicks of a mouse. In both cases, when the Web service is compiled, Caché generates the WSDL descriptor for the service, as well as constructing and formatting the needed “envelope”. Caché also facilitates rapid development by automatically generating a Web page to test the service, without the need to construct a client application.

The Caché Advantage

Easy Connectivity to XML: Caché takes advantage of its capability for multiple inheritance to provide any Caché class a bi-directional interface to XML. The result: Caché classes can easily and quickly be turned into XML documents and schemas. Similarly, XML schemas and documents can be turned into Caché class definitions and objects.

Instant Web Services: Any Caché class can be published as a Web service with just a few clicks of a mouse. Caché automatically generates the WSDL descriptor and the SOAP-formatted envelope.

Secure Web Services: Caché provides features that enable you to easily secure the integrity and confidentiality of Web services. Its use of the WS-policy framework greatly simplifies the task of securing Web services.

Rapid Development of Faster XML Applications: Because Caché’s native multidimensional data structures are a good match to XML documents, there is no need for developers to manually code a "map" that translates between XML and the Caché database. And with less processing overhead, they run faster, too.
Securing Web Services

Web services represent "data in motion" and may come from "unknown" sources. Therefore, they require special security measures.

Caché supports the use of SSL and TLS to secure connections. As for the Web services themselves, information provided with the "envelope" of the service should enable the recipient to verify who the Web service is from, that it has not been tampered with en route, and to decrypt the contents.

Caché implements many features outlined in the WS-Security specifications published by the OASIS-Open standards organization:

- **WS-Security Header**
  - *Provides for message integrity*
    The security header is added to the header of a SOAP message. It contains all the security-related information of the message, including tokens (Username, Binary Security, Timestamp), signature elements, and encrypted key elements.

- **XML Signature List**
  - *Provides for message integrity*
    Enables you to verify who a Web service comes from

- **WS-Security Token Support**
  - **Username Tokens**
    Enables you to verify identity to Web Services that require authentication
  - **Binary Security Tokens**
    Used as references for signature and encrypted key elements

- **XML Encryption**
  - *Provides for message confidentiality*

Configuring security for Web services

To make it easier to configure and manage Web services security, Caché uses the WS-Policy framework outlined by the W3C (World Wide Web Consortium). A wizard is provided that helps application developers generate a WS-Policy efficiently. It includes various options to select the details of the policy.

SOAP Log

It is often useful to log or trace SOAP messages so that you can “see” what was sent over the wire. The Caché SOAP log can (if you wish) capture inbound and outbound messages, as well as security information.
Caché and MultiValue

Caché provides all the capabilities needed to develop and run MultiValue applications (sometimes referred to as Pick-based applications), including the MultiValue:

- **MVBasic language**

- **File access**
  MultiValue files can be accessed by MultiValue programs through the normal OPEN/READ/WRITE MVBasic statements and MultiValue queries. They are also accessible by both MVBasic and other languages through all of the normal Caché mechanisms, including object access, direct multidimensional array access, and SQL.

- **Query language**
  The Caché MultiValue Query Language (CMQL) provides both data selection and report formatting functionality for MultiValue files. Because these queries use Caché’s very high-performance SQL engine, reliability is enhanced, execution is optimized, and a sophisticated set of indexing capabilities can be used.

- **Data dictionary (7- and 10-line structures)**
  A MultiValue file may have a corresponding file description in the MultiValue Data Dictionary, which is directly editable through MVBasic code and through the traditional MultiValue “ED” editor. A MultiValue file may also have a corresponding Caché class definition. A class definition is essential if the data is to be made available for object or SQL access.

- **Procs, Paragraphs and Phrases**

- **Command shell**
  The MultiValue command shell can be run from a terminal environment. In addition to the normal MultiValue command shell capabilities, Caché allows MVBasic commands to be directly executed in the command shell.

This MultiValue functionality is provided as an integral part of Caché – not as a separate MultiValue implementation – and it utilizes the rich Caché multidimensional database engine, runtime functionality, and development technologies. This means that MultiValue users can take full advantage of all the Caché capabilities.
Migrating Applications to Caché

Caché includes features and utilities that can help you migrate existing applications from a number of different technologies to Caché. It allows you to keep your existing data and leverage your existing code, while gaining the performance, scalability, and advanced capabilities of Caché.

Migrating from relational technologies

Caché has specialized implementation of SQL to help you port your applications from:

- **Microsoft SQL Server and Sybase Adaptive Server**
  Caché TSQL is an implementation of Transact-SQL. It supports many of the features of both the Microsoft and Sybase implementations.

- **Informix**
  Caché ISQL is an implementation of Informix ISQL. It supports many Informix ISQL features.

Caché includes system-level classes for importing and compiling DDL, DML, and stored procedures. There is a Data Migration Wizard for importing existing data into Caché.

Migrating from MultiValue technologies

Caché supports migrations from all the popular MultiValue environments, including Universe, Unidata, JBASE, D3, and Reality.

- Caché’s multidimensional arrays (globals) are a superset of MultiValue files, so it is easy to migrate existing data and schema from MultiValue technology to Caché.

- Caché MVBasic is a highly optimized implementation of the Basic programming language used in MultiValue applications. Dialect differences are accommodated by emulation switches – thus minimizing the code changes required when migrating.

- Caché offers CMQL, a MultiValue query language implemented on our highly optimized SQL query engine.

The Caché Advantage

**Easier Migrations:** Utilities and Wizards make migrating from relational or MultiValue technologies as easy as possible.

**Higher Performance:** Caché-based applications have been shown to run SQL up to five times faster than applications based on relational technology.

**More Options for Future Development:** Once migration is complete, you can continue to enhance your applications using the full array of Caché capabilities, including InterSystems Zen technology, object-oriented development, iKnow technology, and more.

**Enhanced Scalability, Security, and Resiliency:** Caché-based applications can take advantage of capabilities such as Enterprise Cache Protocol (ECP), the Caché security model, and mirroring.
Building Rich Web Applications Fast with InterSystems Zen Technology

Many of today’s applications provide a user interface that runs in a Web browser. And many of those interfaces are as rich and sophisticated as those of applications deployed on a desktop computer. Caché includes InterSystems Zen™ technology, which enables the creation of rich Web database applications in record time.

CSP Technology

Underpinning Zen, and providing the mechanisms for running Web applications, is our Caché Server Pages (CSP) technology. Some of the characteristics of Caché Server Pages are:

- **Dynamic Server Pages** – Pages are created dynamically on the application server by application code, rather than having a Web server simply return static HTML. Applications can respond rapidly to a variety of different requests and tailor the resulting pages that get sent back to the browser.

- **Session Model** – All of the processing related to pages from a single browser are considered part of a session – from the first browser request until either the application is completed or a programmable timeout occurs.

- **Server State Preservation** – Within a session, application data on the server – and even the entire application context – can be automatically retained across browser requests, making it much easier to develop and run complex applications.

- **User Authentication** – Any of Caché’s authentication mechanisms may be used.

- **Encryption** – Caché automatically encrypts data in the URL, to help authenticate requests and prevent tampering. The encryption key is kept only on the server, and it is only good for the life of the single session.
Class Architecture of Web Pages

In Caché, every Web page is an object – an instance of a page class that contains methods (code) to generate the page contents. Usually page classes are derived from a standard Web page class "%CSP.Page" that provides every page with various built-in capabilities, such as the generation of headers and encryption. This class architecture makes it easy to change behavior for an entire application and to enforce a common style. It also brings all of the other programming advantages of object programming to Web development.

In general, we recommend that page classes contain only user interface logic. Business logic and database logic should be put into different classes, so that there is a clean separation of user interface code from the business and database logic, and it is easier to add additional user interfaces later.
Zen and Component-Based Web Pages

Zen provides a simple way to rapidly create complex, data-rich Web applications with a sophisticated visual appearance and highly interactive user interface. Zen is not a 4GL – it is a rich library of pre-built object components and development tools based on InterSystems’ CSP and object technology. Zen is especially appropriate for developing a Web version of client/server applications that were originally built with tools such as Visual Basic or PowerBuilder.

Zen components allow much more dynamic interactions – you’re not restricted to the “submit” mechanism to send values to the server. For example, with the Zen form component, you can define your own custom validation, including immediate calls to the server without requiring a page request and subsequent repainting. For users, this represents a more natural way to enter data.

Zen uses the session management mechanism of CSP, providing user authentication, data encryption, and the retention of persistent session data across page requests. All communication between the browser and server occurs by sending objects back and forth using a more sophisticated version of the technique often referred to as AJAX (Asynchronous JavaScript and XML).

Zen-based pages can be easily intermixed with pages developed using other CSP-based approaches to Web development.

What Is a Zen Component?

A Zen component is a class definition that specifies the appearance and behavior of the component on the page. The Zen class definition contains, in a single document, the entire definition of a component, including the style sheets, server code, and client code.

At run time, Zen creates two objects for each component used in a page: a client-side object that Zen automatically creates as a JavaScript object within the browser, and a server-side object. Zen automatically manages the state of both objects and manages the flow of information between them.

Types of Zen Components

The Zen library includes components that implement all the standard HTML control types: input boxes, text boxes, buttons, check boxes, etc. These components have additional behaviors inherited from the Zen control class.

Zen also includes a set of more complex, data-rich components that automatically display data from the database and know how to dynamically update this data in response to user events. For example, Zen’s powerful table component automatically displays data within an HTML table using a database query. The table component supports paging, scrolling, sorting by columns, filtering, and a variety of styles. The contents of the table can be refreshed from the server without repainting the entire page.
Other Zen components include:

- **Menu** – A variety of menu types are supported.
- **Grid** – Add spreadsheet-style behavior to a Web page.
- **Tree** – Display hierarchical data with a tree control.
- **Tab** – A tab component contains a series of tabs, each of which contains a series of other components.
- **Chart** – A rich set of chart components are implemented using SVG, including line, area, bar, pie, hi-low, and XY charts.
- **Graphical Meters** – Speedometers, gauges, etc., let you display data as dynamic visual components.

**Changing the Appearance of the Zen Library Components**

All Zen components support a set of properties that control look and feel. Applications can set these properties at run time to change the values, appearance, and behavior of components.

The visual appearance is also controlled by Standard CSS (Cascading Style Sheet) style definitions. You can override these styles (to change fonts, colors, size, etc.) on an application-wide, a page-wide, or an individual component basis.

You can create sub-classes of the Zen library components to further override appearance and behavior.

**Creating New Zen Components**

One of the main strengths of Zen is that it is easy to create new components.

Every component is implemented as a class. To create a new component: (1) create a new component class that can be a subclass of an existing component; (2) implement a method that renders the HTML content of the component; (3) define the server-side and client-side methods that implement the runtime behavior of the component; and (4) make sure the class includes the CSS-style definitions needed to specify the visual appearance of the component.

**How to Localize a Zen Application for Different Languages**

If desired, Zen automatically maintains a set of all text values (titles, captions, etc.) displayed by an application’s incorporated components in a special localization table. You can export the application’s localization table as an XML document, translate the values to other languages, and import the new tables.

At run time, Zen uses the text values based on the current language preference of the user’s browser.
Chapter Four

The Caché Advantage

Rich Web User Interfaces:
Visually sophisticated, highly interactive pages can be generated that are visually more similar to GUI client/server applications than a traditional simple browser form with a SUBMIT button. The user finds the interactive format more natural and easier to use.

Fast Object-Based Development:
The utilization of pre-built components speeds development and makes it simpler to modify later.

Consistent User Interfaces:
The component-based architecture makes it easier to define and enforce application-wide style and behavior guidelines.

Automatic Generation of Reports:
Reports can be automatically created and printed, or sent to users’ browsers.

SVG Support

SVG (Scalable Vector Graphics) provides a powerful, standard way to display rich graphical data within a Web page. Zen includes the ability to create graphical components that render themselves using SVG and includes a rich set of pre-built SVG-based components.

What Browsers Does Zen Support?

Zen works with Firefox, Chrome, Safari, and Internet Explorer (v7.0 and above). For Internet Explorer, the Adobe SVG plug-in is needed if you wish to use Zen’s SVG components.

Zen Reports

Zen comes with an extensible framework for defining and generating customized reports. A Zen Report is a special kind of Zen class that allows you to specify database reports in XHTML or PDF. Our graphical layout editor gives you complete control over the layout and appearance of reports, and allows the inclusion of (static) Zen charts and graphs. Reports may be output to a browser, or printed, either on the server-side or on a local printer. With Zen reports your Rich Internet Application can exactly replicate the reports users are accustomed to seeing, or easily generate useful new ones.