Moving Your Application from DBF to a Jasmine Object Database
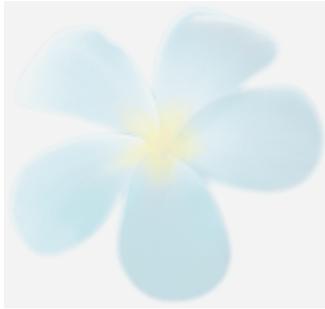
Jean-Paul Bleau
Ordinateurs Laval Inc.

Introduction

Today's computing is tending to move from server-based platforms to the Internet distribution of data. Well-established businesses have the advantage of a wealth of existing data and logic at their disposal. This tremendous asset brings with it the difficult task of integrating this data and logic into new and innovative Internet, intranet, extranet, or client/server solutions. Since older formats like DBF flat files are not very suitable for the Internet, why not move everything to a new type of database: Jasmine ODB? The clash of these generations of data, applications, and technologies creates what we call IT chaos. Successful organizations will be able to mix and match existing systems and data with new technology to create powerful Web-deployed business applications.

This paper shows the differences between the two types of data storage and compares the ways both work. It also shows the implications of using one over the other and describes how to move the data from one to the other in a modular fashion to avoid production conflict and timing. It also explains how to use a middle tier to act as a data server for both the application and the Jasmine server.

The future of application development lies in the deployment of powerful, easy-to-use interfaces. These interfaces present information through intuitive images, sounds, and animation. It is crucial for businesses to offer services and systems that differentiate them from their competitors. This is especially important with respect to e-business applications — if the interface is hard to use, the user will simply go elsewhere. So it is important to port old applications to a new platform to offer what the users want.

Are DBFs Suitable for the Internet?

DBF flat files have been used in the industry for many years now and are widely spread in the entire business world to store data. They are easy to use, easy to maintain and natively supported by many programming platforms. But there are also disadvantages: lack of security, prone to corruption of data in a heavy stress environment, outside the actual standards, and not very client/server ready unless you use a third-party database server. You can still use flat files for storing data that is used on Internet, but you face many limitations. One solution is to use a Jasmine database server and store all your Internet data into classes. In the following pages, you will see the advantages of using Jasmine over DBF for your Internet applications.

What Database to Use?

In the Internet world today, you have two main types of standard databases , relational and object oriented. Remember that OO and relational databases each have their place, and tasks they are better suited to.

In the relational database world we could rely on MS SQL Server or MSDE, both of which are very robust and have all of the networking capabilities built in. Mechanisms exist in these systems to ensure minimal data loss in the event of a crash and to allow multiple users to share the same database. But, how do you do similar networking and transaction saving operations with an OO database as those that come standard with the best of relational database engines?

It helps a lot if you think about the database before you start data design. Unfortunately, OO databases don't have as strong a set of standards as relational databases do. Once you select one, you WILL be bound to it at the hip, and changing vendors will be extraordinarily difficult.

For a start, check out the Web sites offering comparisons, etc. regularly. There are a number of conferences that focus on this stuff, e.g., OOPSLA. There are various books about both OODBs and the Object Database Management Group (ODMG). Quality is variable, and sometimes dated — do a search on Amazon.com first to get a feel for what's available.

To cover some basic comparisons though:

- A query that retrieves a million rows in, say, Oracle will run faster than a query that returns a million objects in Jasmine. Relational databases are optimized for that, pure and simple — it's the ONLY way to get data from them. Relational wins here.

- Relational databases, however, will return the ENTIRE row (or specified fields in their entirety), and a large chunk of the query is sent to the client. Depending on various configurations, Jasmine however will only cache a million Pointers on the SERVER. NOTHING is sent to the client until the client specifically asks for it. So Jasmine wins on a network bandwidth usage. Nice for Internet things and ultra-thin client apps.

- The average relational query involves several joins, each of which slows down the query, requires more indexes to create the join, etc. An object database does NOT use joins — it has inheritance and containing relationships. The latter are when one object contains a reference to another. (For example, a customer record contains a collection of outstanding Invoice objects. Each invoice object contains a list of invoice line items; these in turn contain data for amounts, totals, etc. and a product object, etc.) The OO database only has to return a list of specific customer objects OR invoice objects — it does NOT need to join them, as the client app can navigate through those "containing" relationships as though they were POINTERS.

The upshot is that the more complex the query, the better OO/Jasmine is. The simpler the query, the better Relational is. It's a full gray scale here between black and white. This is why I emphasized the DESIGN aspects. A properly designed relational database tries to be fully normalized, while aiming for the simplest queries possible. A good OO database design focuses on using a few simple queries to get basic data, and then using containing relationships to do ALL other database navigation. As an aside, the OO model is frequently very close to how an OO client application probably looks to the client, and how they use it.

Complex data types (BLOBs, etc.) tend to muddy the waters. Generally, an OO database will manage these more effectively, but exactly how this translates into speed will be more due to your network and server efficiency than the DB model.

There must be a reason why every enterprise is striving to convert their in-house applications over to SQL servers, and software houses are converting their cleverest solutions as well. For the sake of argument, we'll show once more the numerous advantages SQL servers have over DBF or custom file systems developed in-house. The advantages of the classical SQL servers are:

- **Semantic Integrity**

  Primary keys ensure unique identities of records, foreign keys enforce orderly relations, checks and balances ensure valid data entries, etc.

- **Operational Integrity**

  Transaction processing requires that coherent operations upon several tables either succeed or become rejected.

- **Data Backup and Recovery**

  Complete backup solutions help ensure that automatic recovery after server crashes can do their thing.

- **Data Protection**

  No password, no entry. It is as simple as that. I wouldn't want to have to break the encryption of today's databases.

- **Accessibility**

  There is a legion of third-party tools. Report generators, statistics generators, Excel, WinWord; they can all access the data from the SQL server. And composite utilities, such as data mining tools and other data warehouse software, help the end user convert data into information by offering simple, ready-to-use, interfaces in the form of adhoc queries.

- **Productivity**

  SQL, as a development language, is even more productive than the releases of DBF drivers or other proprietary solutions. The developer can delegate to the server a ton of work, which earlier they would have to code for themselves. The maturity of the representative SQL servers on the market today displays itself in the generally eloquent integration of corresponding CASE tools for the visualization and administration of the server.

Differences Between DBF Flat Files and Jasmine

First of all, Jasmine is a database server; this is a major difference. DBF flat files must be handled from beginning to end by your application. This means that your application is also taking care of the index files, locking, commits and so on. This can be accomplished through any kind of link, from a fast server to a thin-client connection. But what happens if the connection breaks? You're in trouble… Unless you use DBF with a third-party database server, like Advantage Database Server by Extend System Inc. or Fortress xBase Server by Loadstone Inc., you have to deal with possible data or index corruptions, because connection interruption is normal… But, if you are looking at Jasmine simply as a way to get more speed out of your existing applications, you may be better off looking at Advantage Server or Fortress xBase Server.

- Jasmine is a database SERVER. If you have never programmed with SQL or any other server product, then it will require a change in thinking and application design, just from that point of view. Most of the DBF quirks and programming/index tricks go right out the door...

- Jasmine is an object database. You cannot and should NEVER just take a set of DBF files and make them classes. Do this and it WILL run like a dog. If you want speed, you need to re-evaluate your data designs and come up with an object design framework.

- A GOOD OODB design is one that does not rely heavily on queries. The speed advantages of OO include the ability to use object references to navigate through your data (with Collection properties making up most of the small lists you need). Queries may be used to find a starting point for work (e.g., locating a Customer), after which you'll be navigating by references.

Other things that you do not have with DBF flat files include, but are not limited to:

- Archiving, backup, journaling.

- Transactions. Support for object and class locking (the OO equivalent of record and table locks). They differ in whether the client can control this, or whether the server takes care of it automatically.

In DBF, you only have to open a file to start browsing the records, editing or appending data. In Jasmine, you need more steps, because the client is no longer controlling the file and data — the server does. We are explaining new terms here:

What Is a Jasmine Session?

An application designed using the Jasmine API can be executed locally on the same machine where the Jasmine server is running or remotely on a different machine. This is done by the client application establishing a connection (or session) to the Jasmine object database (ODB). To start a session, you call odbSesStart()—it is always the first API function that you call. Then, just prior to ending the application, you shut down the session using odbSesEnd()—this is important to ensure that the application disconnects from the server, even if a serious error occurs. Typically, these session management tasks are handled in the main part of your application.

The methods JSession:Init() and JSession:Destroy() are responsible for respectively invoking the odbSesStart() and odbSesEnd() API functions.

The JSession class also provides the following functionality:

- Transaction Management:

    – JSession:TransactionStart() for starting read-only or read/write transactions.

    – JSession:TransactionEnd() for committing or rolling back transactions.

    – JSession:Commit() for committing transactions.

    – JSession:isWithinTransaction() for querying if transactions are currently active.

- ODQL Statement Execution:

    – JSession:Exec(): The function odbExecODQL allows the application to pass a string containing an ODQL statement or ODQL command to be executed in the server. The complete ODQL facility includes data definition, data manipulation, method execution, database query, and a programming language facility. This method can thus be used by a CA-Visual Objects application for augmenting the functionality of the supplied classes with certain features unique to the application and exploiting Jasmine database in a specific manner.

- Error Handling:

    – JSession:CheckStatus(): After each Jasmine API function call, the program should always check the return status. If the return status is ODB_NORMAL or ODB_INFORMATION, then the function has been successfully executed. If the function returns something else, then an exception condition has occurred. In this case the odbGetError function should be called to retrieve information about the exception condition. The odbGetError function returns the error code as well as the error message string. If the buffer area provided to receive the error string is too small, the function truncates the message string before copying it into the buffer. The function also returns the original size of the message string so that the function can be called again with a larger buffer area.

- Getting and Setting ODQL Variables:

    – JSession:GetVar() and JSession:SetVar():ODQL values are distinct from the data values of the CA-Visual Objects Jasmine client application. The odbGetVar() and odbSetVar() functions invoked in the JSession:GetVar() and JSession:SetVar() methods are, respectively, used for transferring data between Jasmine and CA-Visual Objects. These functions use an odbData structure to pass the data back and forth.

    – The Jasmine API provides two sets of functions that perform data type conversion. The set of functions that converts the CA-Visual Objects data to ODQL data has the prefix odbSet, and the set of functions that converts the ODQL data to the CA-Visual Objects data value has the prefix odbGet. In each set, there is one function to work with each of the possible atomic literal types. The odbGet functions are usually used in combination with odbGetVar() to obtain the CA-Visual objects value from the odbData structure. The odbSet functions are used to assign the CA-Visual Objects data value to the odbData structure, which is later passed to the function odbSetVar to be assigned to the ODQL variable or attribute.

- The class provides further functionality for scanning collections and for managing the default connection, including some generalized queries like JSession:GetTopClasses(), JSession:GetAllClasses(), JSession:GetAllClassFamilies(), etc.

What Is a Jasmine Class?

Apart from the fact that Jasmine classes support multiple inheritances, they are pretty much similar to CA-Visual Objects Classes. A class is a collection of objects with common characteristics. Some classes are pre-defined by the Jasmine system; these are included in the class family *systemCF*. These include literal classes such as Integer and String, and classes used by the system such as FamilyManager and Transaction. Additional classes subordinate to the system class *Composite* can be defined by users. These are referred to as *user entity classes*. There are also other classes provided with Jasmine but implemented in the same way as user-defined classes; for example, the multimedia class family is implemented in this way. Information about a class (metadata) is held in a store along with the instances of the class. Classes are organized into hierarchies, representing the natural relationships of specialization and generalization: the hierarchy becomes more specific the lower down you go. For example, Manager is a specialization of Person because every Manager is a Person. The class Manager is referred to as a subclass of the class Person, while Person is a superclass of Manager. Classes have associated characteristics, including *properties* that hold data and *methods* that can be executed. A class inherits the characteristics of its superclasses. The characteristics can also be refined in the subclass and supplemented with additional characteristics.

Properties can be:

- *Attributes*, which are properties whose values are literals or collections of literals, such as a name, a number or a collection of names or numbers.

    Attributes can be:

    – *Instance-level attributes*, which means that each instance of the class and each instance of the subordinate classes has its own value for the attribute.

    – *Class-level attributes*, which means that there is only a single value of the attribute and that this value is shared between the class to which it is defined and all subordinate classes.

    – *Relationships*, which are used to record which one instance or class relates to another instance or class (for example, a person that works in a particular location).

New classes are always based on some existing class; that is, each class has at least one superclass. The classes at the top of a hierarchy of user-defined classes have the system class *Composite* as their superclass. Each class can have more than one superclass, which means that it can inherit the characteristics of more that one superior class. This is referred to as multiple inheritance. You define new classes using the defineClass command. The class must then be built, using the buildClass command. This concept of a Jasmine class is embodied in the *JClass* entity of the Jasmine class library. This class offers, amongst others, the methods for setting and retrieving attribute values (JClass:PutProperty() and JClass:GetProperty(), respectively) , as well as a method for invoking class methods (JClass:CallMethod()). Specific objects derived from the class can be queried using the JClass:FindObjectByProp() and JClass:GetAllObjects() methods. Information about the class (metadata) is retrieved from the Jasmine server and cached in the client application so as to optimize access to this information, which is constantly used by a typical CA-Visual Objects Jasmine application.

What Is a Jasmine Collection?

A collection in Jasmine is an assortment of objects of the same type, such as a collection of numbers or a collection of people. The Collection class is provided to define characteristics common to the various types of collections, including arrays, bags, lists, and sets.

- An *array* is a collection of values with a fixed number of elements defined when the array is first declared. Duplicate elements are allowed, and the collection is ordered.

- A *bag* is a collection of values with a variable number of elements. Duplicate elements are allowed, and the collection is unordered.

- A *list* is a collection of values with a variable number of elements. Duplicate elements are allowed, and the collection is ordered.

- A *set* is a collection of values with a variable number of elements. Duplicate elements are not allowed, and the collection is unordered.

Collections can arise as:

- Attributes within an entity.

- ODQL variables.

- Parameters of methods.

- Return values of methods.

- Members of tuple variables.

- Values of ODQL expressions.

- Query or group expression results.

You can have:

- Collections of atomic literals (for example, List{2, 3, 5, 7, 11, 13}).

- Collections of tuples (typically the result of a query).

- Collections of entities (for example, an array of all project managers in Sydney).

- Collections of classes (for example, a set of all classes in the multimedia class family).

The CA-Visual Objects class JCollection implements the Jasmine collection concept. The implementation, however, aims at achieving a similar navigational scheme as that of Xbase, so as to make access to Jasmine objects from CA-Visual Objects Jasmine client applications compatible to that of existing data sources. The CA-Visual Objects class JCollection, though independent, is mainly used by other classes within the Jasmine classes. The JDataServer class, for example, uses the JCollection class to navigate through all objects of a specific class.

What Is an Object in Jasmine?

A Jasmine object is a thing with its own identity, state, and behavior. In the context of Jasmine, an object is either a literal value such as the number 93.7, or a uniquely identifiable entity in the database often representing an external object or event, such as a person or a project milestone. The class *Object* is the top of the Jasmine class hierarchy, and thus includes *entities* and *literals*. The *Object* system class is the abstract class of all objects managed by Jasmine. There are two subordinate classes of objects:

- *Literal*, which is used for values, such as integers and strings which are permanent and fixed. They cannot be explicitly created and destroyed, and have no identifier other than their value.

- *Entity*, which is used for real-world objects that might have attributes that can change at any time. An entity is an object that has an independent existence in a database. Entities are explicitly created and destroyed, and they have object identifiers (OID) that remain unchanged through their lifetimes. The changing state of an entity is represented at any time by the values of its properties.

Because the Jasmine API is designed for use with an ODB, the type of data that you will be working with will often be an object. The following list of things you can do with objects is short but comprehensive:

- Get the OID of an object into an odbData structure (using odbGetVar(), for example).

- Determine if the value stored in an odbData structure is an OID using odbGetType().

- Get the version number of an object using odbGetObjsVers().

- Execute methods on an object using odbExecODQL() or odbExecMethod().

- Get and set the properties of an object.

- Work directly with the OID as a host language variable.

Attribute values of Jasmine objects are either literals and *single-valued* or a collection of literals (set, bag, list, or array) and *multi-valued*. The CA-Visual Objects class JObject implements the Jasmine object concept. This class offers, amongst others, the methods for setting and retrieving and attribute values (JObject:PutProperty() and JObject:GetProperty(), respectively) , as well as a method for invoking instance methods (JObject:CallMethod()). The CA-Visual Objects class JObject takes care of accessing attribute values of the objects within a Jasmine database server and converting them to/from the respective values of the CA-Visual Objects Jasmine client application variable. Single-valued attributes of the objects are easily mapped onto the respective CA-Visual Objects variables. Currently, the implementation of the CA-Visual Objects Jasmine classes maps multi-valued attribute values (i.e., collection of literals) for Jasmine method parameters requiring collection values and Jasmine methods returning collection values to a corresponding CA-Visual Objects dynamic array value. Jasmine's built-in support for *multimedia data objects* is also reflected in the JObject class through such methods as:

- JObject:IsMMContent() for checking if an object is a multimedia object.

- JObject:MMFile() specifying a multimedia file using the following format: [Major Type of multimedia file,] [Minor Type of multimedia file,] Location and Name of Multimedia file.

- JObject:GetMMDataToFile() for transferring multimedia data from a Jasmine store or an external file on the Jasmine server machine to a specified file on the machine running the client application.

Because a typical Jasmine client application is interested in working with objects in the ODB, many applications accessing Jasmine through the provided Jasmine class will do so mainly with the help of the JObject class. CA-Visual Objects idea of a data server has been incorporated into many existing CA-Visual Objects applications. Those applications need a compatible method of gaining access to data stored in the Jasmine database The CA-Visual Objects class JDataServer encapsulates the required functionalities.

Moving Slowly from DBF to Jasmine

Moving from one type of database to another in one step could be unrealistic in the aspect of real time production data. If you have a system that holds a huge amount of data, you will hardly be able to do the move in one shot. Then, why not isolate the client application from the database server or data source? Yes, this is a good solution. Isolating the client application from the data will make it more open to multiple platforms and more robust. Keeping the client isolated for directly accessing data will also protect your data. One technology you can use to connect your application to the data sources is COM / DCOM. Your application will "ask" for data by sending a request to the COM / DCOM module; the module will then communicate with the data source, parse the data and deliver it to the application. The advantage of this is that the client application knows nothing about the data source. When asking for a customer record, for example, the n-tier (COM / DCOM) module will then take care of the process to use to communicate with the data source. What does that means? It means that you can change the n-tier module anytime when you are ready to change the type of data source. This would be transparent to the client. The following figures show the different layouts:
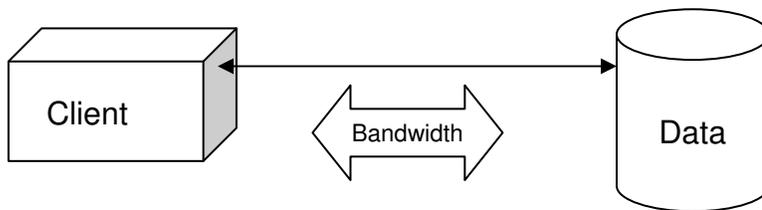
Figure 1. Regular application path.

Using this layout, the client apps have to know the data source and how to manipulate it. The bandwidth is full between data and the application. If a breakdown occurs on the line, this can corrupt data or stop the client application.
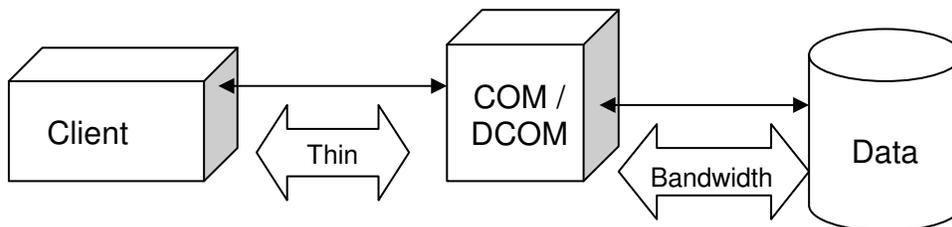
Figure 2. New application using COM / DCOM technology.

Using this layout, the application only needs to know how to query the data. Then the COM / DCOM module will translate the request to access the data source. All of the communication between the data source and the module can be done internally, where communication links are faster and more reliable. The communication link between the client and the module can be a thin connection, because the bandwidth is lower. With good error detection, the client can work transparently with the module. If a communication break occurs, the error module inside the client can redial or try to re-establish the connection and then display error messages for a user-friendly interface. But the link between the module and the data is still alive; and when the module recognizes a failure, it can properly close the data connection and clean the environment. This is a more secure platform.

Later, or anytime you want to upgrade your system, you only have to change the Data part and the communication layer in the COM/ DCOM module. The link between the client and the module will not change. Your client application will not notice the change; it will continue to request data and receive results from the module. If you have ever wanted to change your data server type every six months, for example, you could do it more easily this way.

What this all means is that you can continue to use DBF files during development of the system and slowly move your data from DBF to whatever you want, like Jasmine ODB. So, if you have some legacy applications that use DBF files and are not part of the new architecture, you can continue to use everything until you are at the point where you can safely remove the old applications.

Using Both Types During Transition

It is also possible to use both or many different platforms with the COM / DCOM technology. You just develop what you need inside the COM / DCOM module. Then if a client asks for a record in the invoice table, your module will know that it is still in a DBF file, retrieve it from there, and send it to the client. Afterwards, when the client needs the customer record, the module knows it is in Jasmine and will request for it there, and then return the result to the client application. Now, our COM / DCOM is acting like a real database server. The following figures show the different scenarios:
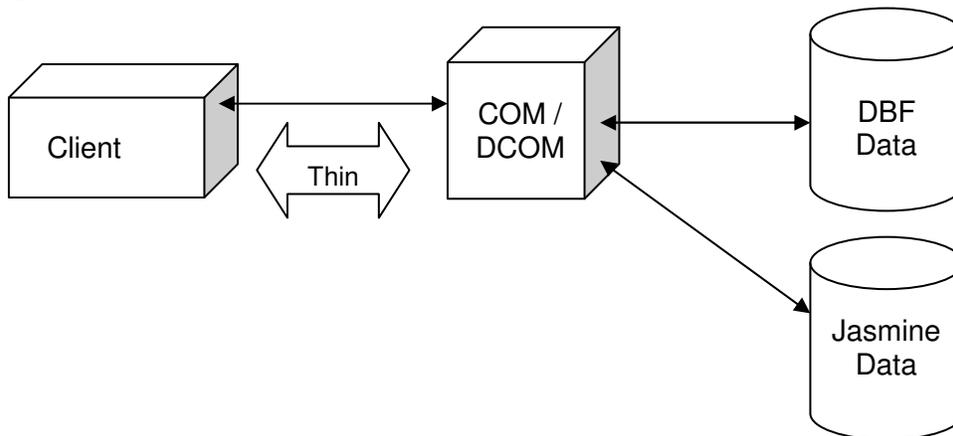
Figure 3. Using both data sources can facilitate the transition.

Other Data Types in Jasmine ODB

Type: ByteSequence
Binary data whose internal structure is unknown or which does not conform to any of the other data types

Handling Abstract Data Types

In some cases, you may want to represent an attribute using an abstract data type where you do not want to make the internal representation visible. Examples of abstract data types that you often encounter are:

- Measures (for example, longitude and latitude).

- Multimedia data types (for example, image and sound).

There are two ways of representing such values in Jasmine ODB:

- **Defining a class**
  You can represent the abstract data type by a class of its own. This is what has been done in the Jasmine multimedia class family described in the *JasmineODB Class Reference Supplement*. The advantages of this approach are that the semantics are very clear in your class definition and that you can hide the detailed representation. The disadvantage is that you need to manage the creation and deletion of these objects: you do not want your database cluttered with objects that are no longer being used.

- **Choosing a literal representation**
  You can decide, for example, that values are to be represented by a string attribute in a particular form. If you do not want to publish the representation, you can provide methods that hide it. For simple attributes, this approach is probably more practical in most cases, though not quite as elegant as defining a class.

The multimedia classes are just that: classes. When you define your own class to hold a bitmap and a Word document, you are actually defining properties to be instances of the Bitmap class and Word class, respectively. Thus, you have to look at the supported properties and methods of those classes to see how to create new instances, add data, etc. This is the same as any other properties, which are objects anywhere else — they can only be assigned a valid instance of that object/class, not a literal property such as a filename. Here I suggest reading the Jascr.hlp file, which documents the class libraries provided with Jasmine.

The multimedia classes basically support class methods for importFile() and instance methods such as exportFile() (import a file, and create an instance of the appropriate class), replace() (for when you want to retain an instance, but change the contained picture file), and sundry others for deleting instances correctly

CA-Visual Objects theoretically should have surfaced the API calls for creating new instances of the multimedia data, extracting and updating them. Check the API definitions for the Jasmine classes — you may find something relevant. Anyone who wants to build something properly will need to learn ODQL and some API stuff if they want to really achieve anything.

Jasmine 'Language': ODQL

Specifically, this "ODQL" acronym means "Object Data Query Language". The name suggests similarities to SQL, the Structured Query Language of conventional relational databases. This, however, only applies to a small portion of ODQL. If you compare the syntax, ODQL is basically a dialect of C++, or actually a preprocessor for this language. It is considered a preprocessor because routines formulated by ODQL that are saved in the database and executed on the server must be compiled by a C++ compiler prior to their use. That is also the reason for the item described in the first issue that, without a C++ compiler, no classes and methods can be saved in the Jasmine database.

At this point I need to make one observation. ALL Jasmine commands (and later the entire ODQL) are case sensitive. Those of you who come from a PC background like me will find this hard to get used to. If you ever get an odd error while using any Jasmine or ODQL command, my first suggestion is check your syntax and CASE. CA-Visual Objects Jasmine classes are implemented pretty good this way. Before you can use SQL classes effectively, you have to know SQL; the same is true for ODQL. So I say ODQL is a must to learn in order to use Jasmine effectively. Learning ODQL is the first and most important step — it's quite a rich language. Once you have this under your belt, a lot of the rest falls into place.

The ODQL interpreter is the command line environment (for the ODQL language), while the Administrator tool is the point-and-click IDE for both the database schema and Jasmine Studio itself. As already mentioned, the syntax of ODQL resembles C++. Developers that are versed with C, and also VO professionals, should not have any problems coping with ODQL. Nevertheless, there are different syntax rules you should consider while working with ODQL. The following rules are mainly valid irrespective of whether you operate with the interpreter or with the editor of the Jasmine Studio.

- All ODQL statements start with the $ placed in front.

- The external ASCII files containing ODQL instructions, which are to be executed on the server, must have a file extension of .ODG. Otherwise, the interpreter displays an error message.

- All statements are terminated with a semicolon (as with C). The semicolon does not play a role in whether an individual statement extends over one or more lines.

- The Jasmine syntax differentiates between upper and lower case. This also applies without exception to all parameters with the interpreter call and is surely unusual for some developers for the first time.

Jasmine includes an object database management system. A Jasmine database stores objects and is manipulated by an object query language, namely ODQL. Contrast this with a relational database, which stores relations (or tables), and is manipulated using a relational query language, typically SQL.

ODQL is used to define the Jasmine schema, that is, the definitions of classes and methods that characterize your particular database. ODQL is also used to access information in the database, with facilities to search for objects matching given criteria, to navigate around the database, and to create and delete objects and update their properties.

ODQL is an object-oriented fourth-generation language. Unlike SQL, ODQL is a complete programming language in its own right. In addition to traditional database operations like select and update, you can declare variables and assign to them the results of expressions. You can also write if statements and while loops. So, in principle, you could write an entire application in ODQL. In practice, however, ODQL is usually mixed with a host language, such as C or C++. The host language will generally be used to write:

- The business logic and user interface modules of applications, using ODQL statements to access data from the database.

- Some of the more complex methods associated with database objects, particularly methods that interact with the external environment or that invoke operating system services.

ODQL can be used in the following ways:

- Embedded ODQL

    ODQL statements are embedded in a host language (C or C++). The embedded ODQL is translated to C or C++ by a preprocessor and is then compiled to object code in the normal way.

- Interpreted ODQL

    ODQL statements are entered by a user at a terminal, to be executed interactively, or are entered from a file. This provides a basic environment for prototyping and testing changes, or for occasional ad hoc query by skilled users.

- C API

    ODQL statements can be executed from within the C API using the function odbExecODQL().

Most ODQL constructs can be used in all these ways. Within the "Reference" section of this help, details of all ODQL features are given. Where there are restrictions for the environments in which those features can be used, this is stated in the relevant topic. The main differences between these environments are:

- ODQL *commands* can be used in interpreted ODQL and by odbExecODQL(), but can only be used in embedded ODQL via the execute statement.

    Examples of ODQL commands are defineClass and compileProcedure.

- ODQL variables declared in embedded ODQL or in a C API application can be associated with variables in the host language (C or C++) to allow the ODQL and host language code to communicate.

    Embedded ODQL statements are prefixed by a $ sign to distinguish them from host language statements

Jasmine Classes in CA-Visual Objects 2.5a

The CA-Visual Objects Jasmine classes are wrappers around calls to the Jasmine API. A lot of the communication through the API takes the form of execution of ODQL statements. It is just as easy to use the Jasmine API from CA-Visual Objects as it is from C. Jasmine has its own API for C/C++ programmers. The classes in CA-Visual Objects 2.5 are actually wrappers around the Jasmine API itself. This is similar to most of CA-Visual Objects, like the Internet classes, which simply provide a better means of using the Internet APIs. Now, the Jasmine API is essentially a means of manipulating ODQL code on the server. (The Jasmine API is basically an Exec() function, with a swag of functions to perform Get/Set operations between C and ODQL/Jasmine data types). CA-Visual Objects is an excellent platform to program Jasmine.

The Jasmine Classes library offers native access to Jasmine, Computer Associates object-oriented, and multimedia databases. Jasmine database objects become native CA-Visual Objects objects inside your application. Method invocations and property sets and gets for the database objects are transparently routed to the Jasmine database server, and intelligent caching is supported inside the library. The library also supports a Jasmine server class similar to the DBServer and SQLTable classes.

Jasmine Classes

JCollection

A collection in Jasmine is an assortment of objects of the same type, such as a collection of numbers or a collection of people. The JCollection class is provided to define characteristics common to the various types of collections, including arrays, bags, lists, and sets.

Collections can arise as:

- Attributes within an entity.
- ODQL variables.
- Parameters of methods.
- Return values of methods.
- Members of tuple variables.
- Values of ODQL expressions.
- Query or group expression results.

You can have:

- Collections of atomic literals (for example, {2, 3, 5, 7, 11, 13}).
- Collections of tuples (typically the result of a query).
- Collections of entities (for example, an array of all project managers in Sydney).
- Collections of classes (for example, a set of all classes in the Multimedia class family).

The elements of a collection must be homogeneous: you cannot mix atomic literals, tuples, entities, and classes in the same collection. Moreover:

- Collections of atomic literals must be the same type of literal (for example, you cannot mix strings and integers).

- Collections of tuples must be compatible (for the definition of *compatible*, see the Jasmine documentation for further information about tuples).

- Collections of classes must be subordinate to some common user-defined class (for example, a collection of all of the subclasses of Person).

- Collections of entities must belong to the same user-defined class, including instances of its subordinate classes (for example, a collection of Persons can include Managers because Manager is a subclass of Person).

In the previous version of the Jasmine classes, all collection attributes were handled on the client side of application (i.e., in CA-Visual Objects code) using the array data structure; collections were automatically converted to arrays.

In version 2.5a this default behavior is maintained. However, you now have the option of creating true collections where the collection object identifier is the first and only component the client application knows about in that collection. The client application can then query the Jasmine database server for subsequent components of the collection (like the number of elements in the collection or a definite element within that collection, say the last element, etc.). This feature reduces the traffic overhead between the client and server components of the application when handling large collections. The default behavior, however, is quite adequate for handling small collections (denoting, for example, the address of a customer, etc.).

To turn this new behavior on, call the function:

```
SetCollectionConceptFlag(TRUE)
```

The Boolean parameter TRUE indicates to the Jasmine classes that all collections created henceforth should not be treated as arrays when they or their elements are being queried.

Note, however, that you cannot mix the two methods of handling collections; i.e., creating a collection to be retrieved as an array and subsequently trying to retrieve it as a collection. Collection traversal is achieved using the class methods and accesses, JCollection:Advance(), JCollection:CurrentItem, etc.

JClass

Provides a class from which proxy objects can be instantiated in Jasmine client applications. These proxy objects correspond to objects representing user-defined classes in a Jasmine database. An object derived from this class contains information about a Jasmine database class. A number of properties and methods are available for respectively accessing and manipulating that information.

Proxy Class Specification

To create proxy class objects, one is not limited to the class JClass. It is possible to subclass from the JClass and then to create classes using the subclassed class. The Jasmine Classes library, therefore, has to know the class that the application (or a section of the application) is using for creating proxy classes. Using the VOProxyClass access/assign you can set and/or retrieve the symbol of that proxy class defining class.

Instantiating proxy objects should be done using the CallJClassInit() function instead of using the Init() method of JClass (i.e., using JClass{}). If no other symbol is specified for creating proxy classes, the default behavior is to use JClass.

JDataServer

Manages a Jasmine collection of objects selected from a specific Jasmine class from which a group of properties is chosen. Once a scan is opened on a Jasmine class, the elements of the collection can be accessed using the methods of the JDataServer class.

Restricting the Default Collection Automatically Associated with a JDataServer

In the previous version of the Jasmine classes, all collection elements associated with the specified Jasmine class are retrieved from the database server to the client application during the instantiation of a JDataServer object. The current version of the classes now permits you to restrict the set of the initial collection. The sWhereClause parameter of the Init() method enables you to do just that. That parameter represents a syntactically correct ODQL string. All attributes of specific Jasmine classes have to be prefixed either with a "<classname>." or with an "x."

For example, to select only Garments of style "Coats", you would write:

```
JDataServer{goSession, "CAStore::Garment", "x.style == "+_CHR(34)+"Coats"+_CHR(34)}
```

Related Issues

The JClass:FindPropertyByName() and JClass:GetAllObjects() also use the specified "where" clause in filtering objects from the Jasmine database server. There is an Access/Assign "Where" in JClass for manipulating this property.

JSession

Provides a class for managing Jasmine sessions. A session must be started before any objects managed in the Jasmine database can be manipulated. When a session is started, the name of the client environment file to be used for the session is nominated. The same application can be run in different environments, consequently, by using different client environment files each time.

New methods

JSession:GetObjsVers(nOfOid, asOid)

Retrieves the versions of a Jasmine database object. This method can be used for handling concurrent accesses to Jasmine objects. The first parameter specifies the number of elements to be analyzed. The second parameter is an array with the OIDs of the elements whose versions are to be retrieved.

JSession:AddProperty()

JSession:RemoveProperty()

JSession:ReplacePropDefault()

JSession:GetJObjectCName()

Utility methods for manipulating attributes of Jasmine database objects (see the respective Jasmine documentation for use of the attributes).

JSession:IsSubClass(sCFName, sClassNameA, sClassNameB, xRecursive)

Optionally checks recursively if the class specified by sClassNameB is a subclass of the class specified by sClassNameA within the Jasmine ClassFamily specified by sCFName.

Conclusion

Writing applications for the Internet can be a big challenge, but the rewards will be worth the effort. Most users will be very happy to work with their favorite application from the Internet, and they will promote your product for you. With a good organization, a good architecture, good tools like CA-Visual Objects®, Jasmine® ODB, and Jasmine® ii, and teamwork between the end users and the programmers, you will make your product the best on the market. I strongly recommend that you subscribe to the SDT International magazine. It contains very good articles on Jasmine and CA-Visual Objects; the publisher can be reached at http://www.vocager.de.

There are also tools to help you make the transition. One of them is SP-JasmineDBF by Virtiron Inc. at www.virtiron.com. Here is a quick description:

Bridging the gap between DBF and the Jasmine Server, this product takes existing .DBF files and generates the appropriate Jasmine classes and objects:

- Allows for .DBF files to map to classes patterned on the DBServer class OR allows structure to be used as CLASS structure.

- Allows for additional properties to be added to classes as the DBF structure is incorporated.

- Creates objects from records in a .DBF file.

- Full documentation generated by SP-DOCAPP 1.0 and INV.

- Each module sold separately to developers.

- License fees based on server access.

- Full documentation generated by SP-DOCAPP 1.0.

Bibliography

CA-Visual Objects® 2.5, printed and online documentation. Computer Associates.

Jasmine® ii Brochure. Computer Associates.

Microsoft Knowledge Base. Microsoft Corporation®.

Microsoft Windows NT Resource Kit®. Microsoft Corporation®.

Microsoft Win 32 API documentation. Microsoft Corporation®.

Discussions on various newsgroups.

Documents from different previous CA-World sessions

SDT International, The Magazine for C/S and Internet Programming

Biography

Jean-Paul Bleau is a software developer and consultant. He has been working with computers since 1978, with dBase II when it was on CP/M, with CA-Clipper® since 1986 and CA-Visual Objects® since the pre-release in 1994. He built specialized software for Fertility Centers to take complete care of electronic patients' charts in a health care environment, and he created CGPlus, a complete accounting system for CA-Visual Objects® 2.5 and CA-Clipper®. He is also the author of several utilities that share data with accounting packages like Point-of-Sales written in CA-Clipper®. Jean-Paul has worked in the US, Belgium, France, Canada and Africa, where he has had to deal with different types of taxes and accounting cultures. He has been attending CA-World since 1995. He was a speaker at many CA-World Technicon sessions and CA-World's eBusiness Solutions in Internet Time. He is also available to speak at other conferences or for corporate training. Jean-Paul can be reached by email at JPBleau@qc.aira.com

Copyright Notice

Trademark Acknowledgements