



# INTRODUCTION TO **CA Jasmine ii ODB**

For Microsoft Windows XP Professional  
Revised January 2007

<b>1.    </b>	<b><i>Introduction</i></b>	<b>3</b>
1.1	Remarks about this tutorial and last minute changes	3
1.2	Some Reading Guidelines...	3
<b>2</b>	<b><i>Overview of the Jasmine ODB</i></b>	<b>4</b>
2.1	Stores, Extents and Class families	4
2.2	Jasmine Studio versus the ODQL Interpreter	5
<b>3</b>	<b><i>Setting up the Jasmine ODB environment</i></b>	<b>7</b>
3.1	Specifying the remote server	7
3.2	Connect to the local Jasmine installation	9
<b>4</b>	<b><i>Getting started with the Jasmine Tools and ODQL</i></b>	<b>10</b>
4.1	Jasmine Studio	10
4.2	Using the ODQL interpreter (CODQLIE)	12
4.3	ODQL	14
4.3.1	Writing an ODQL query	14
4.3.2	Using collections	15
4.3.3	Additional useful methods	16
<b>5</b>	<b><i>Modifying a Jasmine Database</i></b>	<b>19</b>
5.1	Creating Classes	19
5.2	Creating Attributes	21
5.3	Creating Instances of a Class and Adding Values to Attributes	23
<b>6</b>	<b><i>Creating a Copy of the CAStore Database</i></b>	<b>26</b>
<b>7</b>	<b><i>References</i></b>	<b>28</b>

## 1. Introduction

Jasmine™ from Computer Associates (CA) is a large object-based eBusiness platform with numerous tools and services, ranging from business process identification, object-oriented database and application development, to dynamic report generation, support on business knowledge and multimedia management. This introduction focuses on the Jasmine Object Database (ODB), leaving much of the eBusiness aspects unmentioned.

The powerful ODB approach enables us to think about and model our domain in real-world terms without force-fitting data into traditional tables, rows, and columns as in traditional relational databases. Jasmine stores everything - entities, characters, numbers, pictures, video, and audio as objects. The benefits with this are numerous, inheritance, dependencies and querying is made easy through the use of fully object-oriented concepts and the *Object Database Query Language* (ODQL).

It's even possible to map your own database models directly into your programming language of choice (for example Java, C++, Visual Basic etc), meaning that the DBMS is able to use database models to generate application basics. Even the opposite is possible; you could just as easily turn your application data layer into a database model in Jasmine. Naturally this speeds up the development process rapidly since the effort to transform conceptual models into database models is reduced. When the DBMS produces the foundation of the application code it also lessens the gap between the application and the database.

The purpose of this document is to familiarize and present common tasks and concepts found in the Jasmine ODB environment. The aim is not to provide a complete and in-depth description of the whole eBusiness enterprise but rather to point out key aspects that are helpful to students at the Department of Computer and System Sciences (DSV) at the University of Stockholm / Royal Institute of Technology (KTH) in Sweden when performing different course assignments. A complete and detailed documentation of Jasmine is available under *Start » Programs » Databases » Jasmine ii » Documentation*.

### 1.1 Remarks about this tutorial and last minute changes

This tutorial's sole purpose is to make Jasmine as easy, fun and comprehensible as possible. With this in mind we also know that this introduction is far from complete and that it only targets parts of the whole DBMS environment. Vise from experience we know that errors are going to be introduced by this text, ranging all from typos to misunderstandings, so therefore it's recommended to keep an open mind to what is presented. ***Please do not try to skip through the text since that increases the risk of misinterpretation dramatically. Try to read as much as possible!***

### 1.2 Some Reading Guidelines...



**Actions:** This icon symbolizes operations you are expected to perform by yourself. This can for example be typed commands or mouse clicks on user interface buttons.



**Recommendations:** This icon represents additional tips that are not required but could help speed things up as you go along.



**Warnings:** These symbols point out things to avoid and well-known sources of errors.



Stores are made up of *Extents* that are actually files on the physical drives. Thus by using multiple extents, a store can span several drives and be as large as your operating system permits. Optimization is achieved by placing stores for different applications on different physical drives.

Since Jasmine is object-oriented, classes and class instances (objects) are a vital concern of the DBMS. Related classes and their instances form *Class Families*. A store can hold several uniquely named class families. For more information refer to the *Jasmine ODB – Database Design and Implementation* manual found under **Start » Programs » Databases » Jasmine ii » Documentation**.

## **2.2 Jasmine Studio versus the ODQL Interpreter**

Jasmine commands and utilities have basically two major purposes:

First the *Operating system commands* (programs) that act on the database's physical file storage: backup and restore, allocating file storage, defining class families, reading and defining configuration settings, starting and stopping the database engine, connecting to the database (from a client), etc. These commands are commonly used to install and configure the DBMS and are not often used for educational purposes.

Secondly, the *Database commands* and methods that act at the class and instance level, on items stored inside the database. They include the *Object Definition Language* (ODL), *Object Manipulation Language* (OML) and *Object Query Language* (OQL). These commands can only be executed within Jasmine methods, i.e. the *ODQL interpreter* or Jasmine client programs like the *Jasmine Studio*.

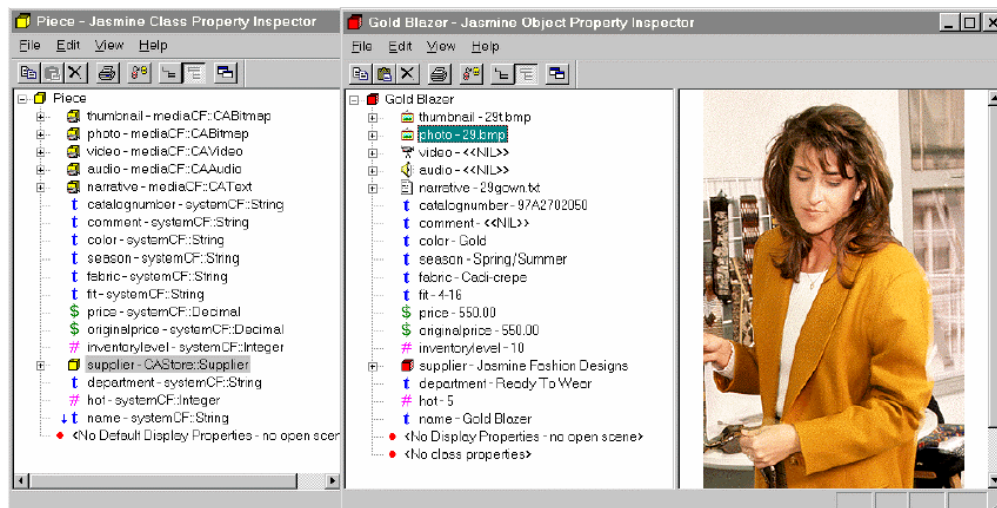
In chapter 0 we'll set up the Jasmine DBMS environment, while chapters 4 and 5 are dedicated to database functionality and commands.

If we want to work with our database, we have two main tools in our disposal:

- A graphical user interface called *Jasmine Studio* comprised of the *Jasmine Application Manager*, the *Jasmine Class Browser*, and some more tools.
- The *ODQL interpreter*, a command-line tool.

Choosing the correct tool for each situation is very important, and impacts on your ability to administer, migrate, upgrade and support your database. The *Jasmine Studio* is a point and click graphical user interface tool for creating applications and for browsing the database schemas and their contents (the objects) as well as for changing the database structure. We will only use the Jasmine Class Browser (for working with the database). The advantages of this tool are:

- It's very easy to learn and use.
- Only practical way to view multimedia data instances.
- Drag and Drop is convenient way of populating data, especially multimedia types.
- Building classes and compiling methods is a one-touch operation.



The *ODQL interpreter* is the antithesis of the Studio. It is a command line environment which only accepts ODQL language input. While this makes the interface hostile to the casual user, it's a very powerful tool. Consider it for the following:

- It's the only way to run ODQL scripts. These could be testing code for methods, creating instances of data, or generally automating some task.
- It's the only medium which supports the bulk creation of test data (apart from using the LOAD utility, which assume you already have the database created on another server).
- It supports detailed inspection of all class attributes, instance data, etc.
- Classes defined here can select the type of collection properties they use (Array, List, Set or Bag). The Database Administrator does not allow this for some reason.
- You can run ANY type of test query, including the *Group By* construct which the Studio does not support at all.
- Use the ODQL Class to help you build queries, and the Tuple structures required to hold the results. This can get quite messy, and the ODQL class is a great help when writing methods.
- You can run test scripts and capture the output to text files, thus offering a way of documenting/validating work done.

```

C:\>Command Prompt - codqlie

D:\>\codqlie
Client ODQL Interpreter
Jasmine ii ODB
Portions of this product Copyright 1996-2000 Computer Associates International,
Inc.
Portions of this product Copyright 1996-2000 FUJITSU LIMITED
Portions of this product Copyright 1996-2000 Computer Associates International,
Inc. & FUJITSU LIMITED

Connecting to host IS008.
IS008(systemCF) > _

```

As you notice, the interpreter is used for two things: handling complex operations, and writing/running ODQL scripts. It should also be noted that the tool is unforgiving when it comes to errors. Minor errors in case can cause long chains of messages to appear and errors in methods can be awkward to track down. Complexity aside, those of you who are already familiar with other DBMSs will soon see the benefits of the ODQL interpreter.



*Being able to record all database schema information, schema changes, compilation instructions and even data unloading and loading into scripts which can be executed, and re-executed at will is a vital part of database administration. If you lose a server and have to reconstruct the database from a particular point in time, you need full logs of all changes, and scripts to run to bring it up to par quickly. This is definitely not something you use a point-and-click tool for.*

Finally, the documentation of Jasmine is a great source of information and examples of how to do most things. Check out the *Jasmine Database Developer's reference* as well as other help files found under **Start » Programs » Databases » Jasmine ii » Documentation**.

### 3 Setting up the Jasmine ODB environment

The following section addresses steps necessary to run. *These steps are VERY important since the default Jasmine setting must be configured locally on each computer in order for the DBMS to run properly!* All this necessary configuration is already done on the prepared disks, but it is presented here for your information.

*You only need to do the configuration presented in sections 3.1 and 3.2 once. Once the environment is properly set up, no further configuration of the Jasmine ODB should be needed.*

#### 3.1 Specifying the remote server



Log on to the computer with your database account to get database administrator privileges. Use the account information you received during account registration/disk checkout in the beginning of this course.



Open a *Command Prompt* window



In the *Command Prompt* window that opens, type in the command: **hostname** to retrieve your computer name.



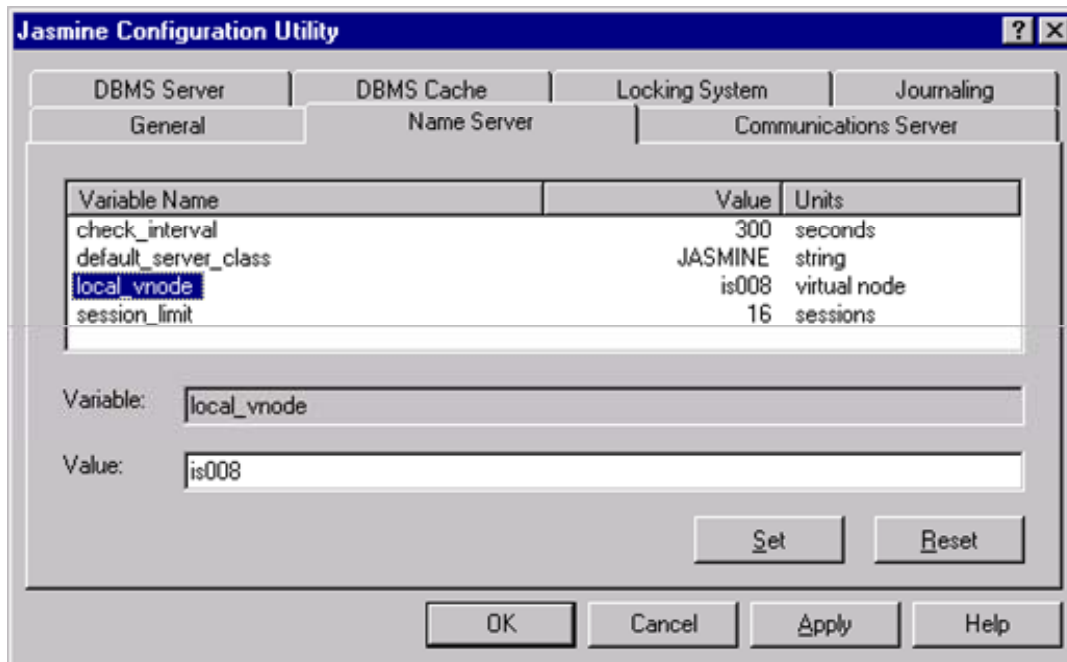
**is008** (this is the default hostname of the prepared disks)





Start the *Jasmine Configuration* application found at the start-menu path: **Start » Programs » Databases » Jasmine ii » Jasmine Configuration**




In the *Jasmine Configuration Utility* window, select the *Name Server* tab and change the parameter *local\_vnode* to the name you received from the hostname command previously (in this example **is008**). Mark the *local\_vnode* row and type in your hostname in the *Value* field. Then press *Set* to commit your changes and then *OK* to close the window.





 Stop the Jasmine server by typing in the command: **net stop Jasmine** in a *Command Prompt*.


 **C:\Documents and Settings\db2admin>net stop Jasmine**  
**The Jasmine service is stopping.**  
**The Jasmine service was stopped successfully.**

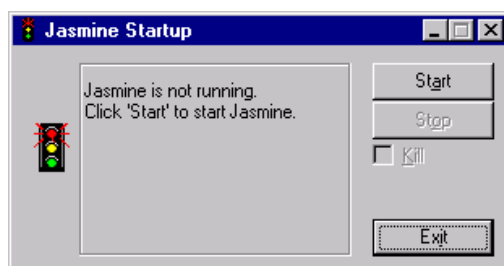
Or, if you don't have the Jasmine service started you get:

 **C:\Documents and Settings\db2admin>net stop Jasmine**  
**The Jasmine service is not started.**  
**More help is available by typing NET HELPMSG 3521.**

 Restart Jasmine with the *Command Prompt* command: **net start Jasmine**

 **C:\Documents and Settings\db2admin>net start Jasmine**  
**The Jasmine service is starting..**  
**The Jasmine service was started successfully.**

 You can also use the graphical tool *Jasmine Startup* (found under **Start » Programs » Databases » Jasmine ii » Jasmine Startup**) in order to easier start and stop Jasmine.





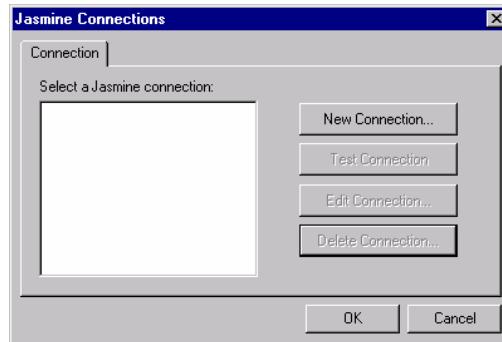
### 3.2 Connect to the local Jasmine installation



After the remote Jasmine server is set up on your local machine, start up *Jasmine Studio* by using the start-menu path: **Start » Programs » Databases » Jasmine ii » Tools » Jasmine Studio**



In the *Jasmine Connections* window that appears, press the *New Connection...* button.

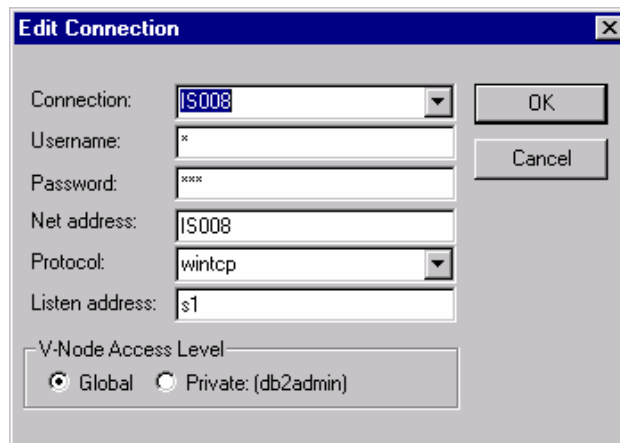


NOTE: If you already find a connection use the "Edit Connection" button to change its parameters.



In the *New Connection* window, change the parameters as follows:

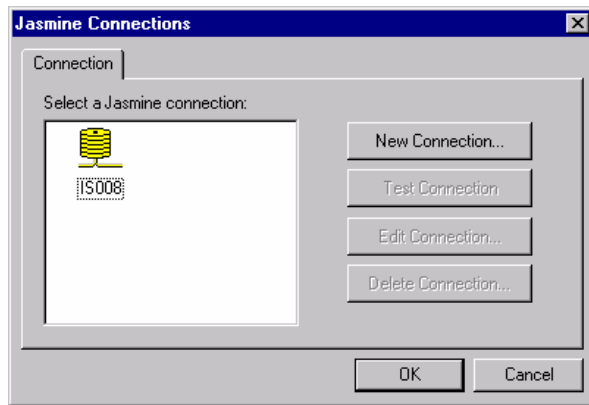
**Connection:**                **is008** (the same hostname as you got earlier)  
**Username:**                **\***  
**Password:**                **is4**  
**Net address:**            **is008** (the same hostname as you got earlier)  
**Protocol:**                **wintcp**  
**Listen address:**        **s1**  
**V-Node Access Level:**    **Global**



Press *OK* to create a new connection to your local Jasmine server and to terminate the *Add Connection* window.



From now on you can use the existing local connection icon in the *Jasmine Connections* window to access your databases. Double-click it to use it.





## 4 Getting started with the Jasmine Tools and ODQL

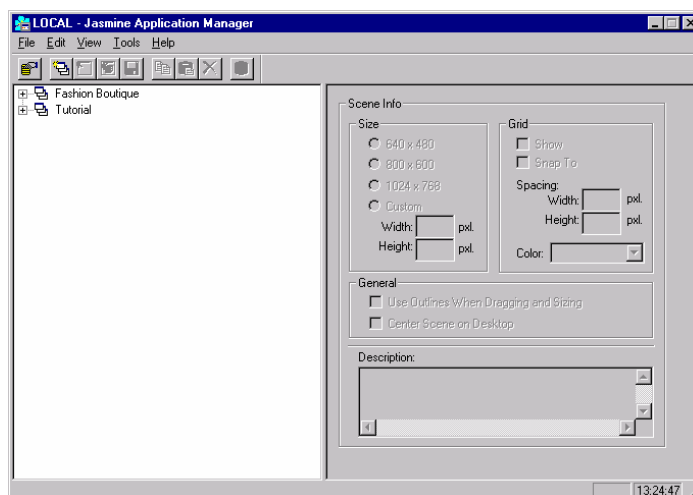
This chapter presents a short description on how to perform database commands through *Jasmine Studio* and the ODQL interpreter. It also gives a short introduction to the Jasmine ODB specific ODQL language.


### 4.1 Jasmine Studio

This section covers how to start up and work with the graphical user interfaces in Jasmine Studio, the *Jasmine Application Manager* and *Jasmine Class Browser*.

 Start Jasmine Studio by following the start-menu path: **Start » Programs » Databases » Jasmine ii » Tools » Jasmine Studio**

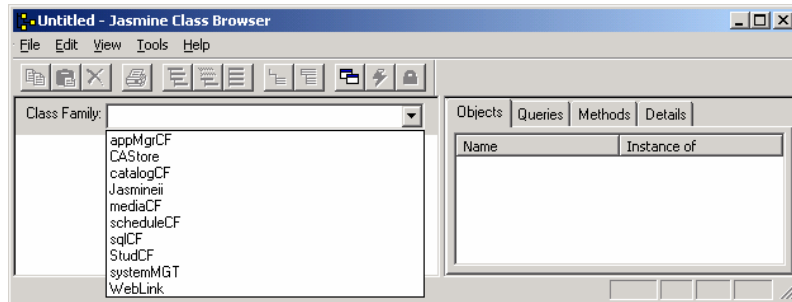
 Double-click the local connection icon in the *Jasmine Connections* window to bring up the *Jasmine Application Manager*. This tool is used to create applications from within Jasmine through the graphical user interface. As you can see there are two examples of applications available: *Fashion Boutique* and *Tutorial*. If you are interested, please feel free to get familiar with the examples and the functionality of the *Application Manager*; however this tutorial does not cover any of its functionalities.



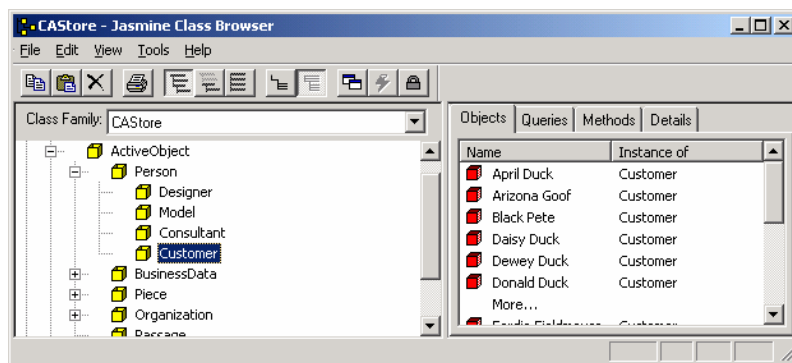
 In the *Jasmine Application Manager*, choose: **File » Database Administration...** from the menu-bar to start the *Jasmine Class Browser*. The *Class Browser* is used to visualize and manipulate user data.



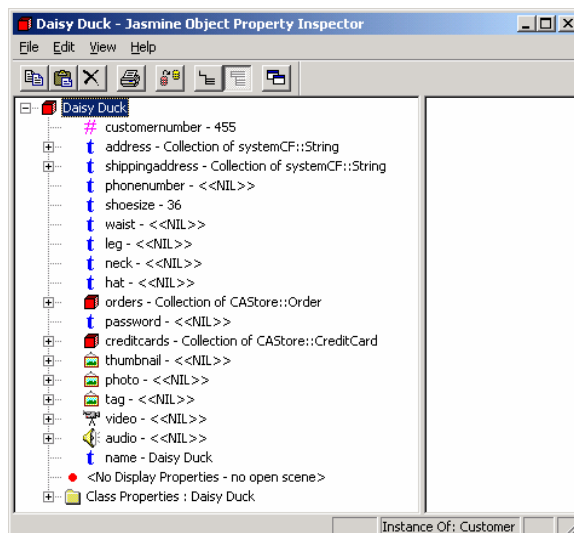
To access the data, select your *Class Family* from the drop down list and note that all the classes are shown in the hierarchy to the left as yellow boxes.



Select the class you want by clicking on its yellow icon and all instances of the class are shown on the right hand side as red boxes.



Double-clicking on a red boxed instance brings up the *Jasmine Object Property Inspector* where the values of the specific instance can be viewed and changed. Note that objects can in turn consist of other objects, both as collections and as singletons. These objects are also represented as red boxes within your selected object.



## 4.2 Using the ODQL interpreter (CODQLIE)

ODQL is the querying language used in Jasmine ODB. It is really more of a programming language that's fully capable of sequences, selections and loops. It can be used directly from a command line environment through the ODQL interpreter named *codqlie* or embedded in a programming language similar to embedded SQL. Using embedded ODQL is outside of the scope of this introduction and the following steps just show how to start and stop *codqlie* and issue ODQL queries both interactively and as script through this tool.



***The Jasmine server must be running in order for the codqlie tool to work. Use the net start Jasmine command in a Command Prompt to check this.***



To start *codqlie* in *interactive mode*, start a *Command Prompt* window and issue the command: **codqlie**



```

D:\>codqlie
Client ODQL Interpreter
Jasmine ii ODB
Portions of this product Copyright 1996-2000 Computer Associates International, Inc.
Portions of this product Copyright 1996-2000 FUJITSU LIMITED
Portions of this product Copyright 1996-2000 Computer Associates International, Inc. & FUJITSU LIMITED
Connecting to host IS008.
IS008(systemCF) > _

```



Issue an ODQL command by typing in your query row by row, ending each line with the character “;”. Execute the row by pressing return. (User input in bold):

```

D:\>codqlie
Client ODQL Interpreter
Jasmine ii ODB
Portions of this product Copyright 1996-2000 Computer Associates International, Inc.
Portions of this product Copyright 1996-2000 FUJITSU LIMITED
Portions of this product Copyright 1996-2000 Computer Associates International, Inc. & FUJITSU LIMITED
Connecting to host IS008.
IS008(systemCF) > defaultCF CStore;
IS008 (CStore) > OrderItem set ois, oi;
IS008 (CStore) > ois = select OrderItem from OrderItem;
IS008 (CStore) > scan(ois,oi){oi.name.print();};

```

```
Apple Red Windbreaker
Beige Bag
Beige Crocheted Vest
Black Satin Sandal
Black Satin Sandal
Miniature Bag
White Pantsuit
Red Bag
...
IS008 (CAStore) >
```



To quit the *codqlie* shell, type the command: **end;**



Another way of issuing commands is through scripts. Specify your whole ODQL query in a text file and place it on a drive. Process the script by giving the text file as a *-execFile* parameter to *codqlie*. Type in: *codqlie -execFile <drive:\directory\filename>* in a *Command Prompt* to execute the script. In this example we execute the file *test.odql* placed at *d:\myQueries* by the command: **codqlie -execFile d:\myQueries\test.odql**



#### Client ODQL Interpreter

Jasmine ii ODB

Portions of this product Copyright 1996-2000

Computer Associates International, Inc.

Portions of this product Copyright 1996-2000 FUJITSU  
LIMITED

Portions of this product Copyright 1996-2000

Computer Associates International, Inc. & FUJITSU  
LIMITED

Connecting to host IS008.

Apple Red Windbreaker

Beige Bag

Beige Crocheted Vest

Black Satin Sandal

...



A good thing is to use batch-files in Windows to automate processes. A batch-file is just like a text file with one command on a single row. Create a new text document and type in your commands into it (for example just **codqlie**) and save it as *run.bat*. Next, give the command: **run** to execute every command you entered in the batch-file. (In this case it will just start *codqlie*). Use this technique to perform many commands at once but remember to issue the *run* command in the same location as you placed the batch-file.

The example below creates a folder named *myTest* on the *d:* drive, copies the file *myQuery.odql* to this location and executes *codqlie* with the file as a value given to the parameter *-execFile*. The results are piped out to the file *results.txt* which in turn is opened for editing through notepad.

```

@echo off
rem This text is treated as a remark
echo Creating folder...
md d:\myTest
echo Copying file...
copy c:\document\myQuery.odql d:\myTest
echo Executing codqlie...
codqlie -execFile d:\myTest\myQuery.odql >
results.txt
echo Starting notepad...
notepad results.txt
echo End of script!

```

### 4.3 ODQL

This section presents an introduction to syntax and methods necessary to develop useful Jasmine ODQL queries.

The easiest way to run ODQL queries against Jasmine ODB is through the ODQL interpreter (codqlie) from a *Command Prompt window*, as mentioned in section 4.2. Codqlie takes a number of parameters where the most important ought to be *-execFile*. This parameter runs the file specified as a script against the database. Other useful parameters are easily displayed through the *-h* parameter which displays the full argument list.



All commands issued in codqlie have to have an ending semicolon “;”.



It is possible to commentary text right in the middle of the ODQL text. Use the */\** markup to indicate the start of a comment, and *\*/* to terminate it.



All commands in ODQL are case sensitive, so check spelling and names carefully since Jasmine has a poor error feedback. Many of the reserved words are spelled in lower case, check the correct syntax in *Jasmine Database Developer's reference* found under Start >> Programs >> Databases >> Jasmine ii >> Documentation or at [ftp.ca.com/pub/jasmine/docs/nt\\_202](http://ftp.ca.com/pub/jasmine/docs/nt_202).



***ODQL is not OQL! OQL is the foundation for ODQL but the syntax and functionality differ quite a bit. (For example ODQL cannot traverse through collections with the ‘.’ qualifier. These need to be scanned through). In return you are able to use IF-THEN-ELSE statements etc. Do not expect every OQL syntax to work in Jasmine!***

#### 4.3.1 Writing an ODQL query

Every ODQL query acts within a class family and this is the first thing that needs to be specified when creating a query. (For a description of class families, refer to section 2.1). If you do not change the class family, the default family *systemCF* will be used instead, effectively disallowing you access to any user defined data. To change the class family, use the command *defaultCF <class\_family\_name>* to specify what class family that are of interest. In the following example we access the user defined class family *CAStore*:



Start a *codqlie* session by opening a *Command Prompt* and issue the command: **codqlie**



Type in: **defaultCF CStore;**  
...

Next thing we need to do is to specify variables to be used during the processing of the query. This includes *Bags*, *Lists* and *Sets* as well as *Integers*, *Strings*, *Reals* and *Dates*. Bags (unordered collections allowing duplicates), Lists (ordered collections allowing duplicates) and Sets (unordered collections without duplicates) are *collection classes*, meaning that they could consist of any number of variables of a given type.

In contrast, singletons are single data items of specific types. This means for example that a bag could consist of a number of string variables but a singleton could never consist of anything else than a value. To make an example we define a select query, retrieving only one shoe size (*a string singleton NOT a collection*) and put the result in our specified string variable named *str*:



```
...
String str;
str =      select Customer.shoesize
from Customer
where Customer.name == "Donald Duck";
...
```

This only works because the query returns a single value and not a collection of values.

Finally, when we have the result, we need to present the information to the user. This can be done through the *print* method which is inherited to all system-defined variables. Specify the following command to print out the content of our *str* variable and end the query by specifying the *end* command:



```
...
str.print();
end;
```



42

### 4.3.2 Using collections

If you want to work with collections, things get a little more complicated. The following example retrieves a collection of objects. We begin by specifying the class family and then declare our collection by using the syntax *<data\_type> set <collection\_name>*. We also specify an order item variable named *order\_item* that we are going to use later on. The first part *data\_type* defines what kind of data the collection shall hold. The argument *collection\_name* is the handle to this object and the keyword *set* is used to specify that the variable is a collection.




```
defaultCF CStore;
OrderItem set order_item_collection;
OrderItem order_item;
...
```




It is possible to declare several variables of the same type on a single row. The following example expresses the same as the rows above:

```
defaultCF CASTore;
OrderItem set order_item_collection, order_item;
...
```

This creates an unordered collection of order items named *order\_item\_collection*. So far, this is an empty collection since we haven't allocated any order items to it yet. To do this we add the select statement:


```
 ...
order_item_collection = select OrderItem from
OrderItem;
...
```

This will put all order item instances from the database and put it in our collection. To print the collection's content we can use the same *print* method as before but there is a better way. By using the ODQL method *scan(<collection>,<singleton-handle>){}* we make the printout look nicer. The scan method takes two arguments, a collection to scan and a singleton handle of the same data type as the content of the collection. This is why we needed to declare the singleton variable *order\_item* earlier:

```
 ...
scan(order_item_collection,order_item){
    order_item.name.print();
};
end;
```

```
 Apple Red Windbreaker, Beige Bag... (etc.)
```


This would go through the whole bag of order items and, for each loop, retrieve the order item attribute *name* and print this to the user. The singleton *order\_item* acts as a handle for the active attribute in the *order\_item\_collection* during the scan loop.

 ***Remember that stepping through data with the '.' qualifier only works on one-to-one relations. As soon as you go from a single object to a collection (one-to-many) you have to use some sort of iteration to get the data.***

#### 4.3.3 Additional useful methods

There are several more useful methods in ODQL that save a lot of workarounds. In this section we present a few of them. To find other methods useful to you, take a look at the Jasmine Database Developer's Reference.

The first method is the *hasElement* method. This only works on collections and scans through them to find matches of the given argument. If the object is found, the method returns true. If not, it returns false. The following example searches for any accessory that is not colored blue. It's important only to search for object of the same data type as of the objects in the collection.

```
 /* Show only non-blue accessories */
defaultCF CASTore;
String set strset;
strset = select Accessory.name
```



```
from Accessory
where not(Accessory.colors.hasElement("blue"));
strset.print();
```



```
Bag{ Red Umbrella, Beige Bag, Pigskin Bag, Red Bag,
Large bag, Red Sun Hat, Black Satin Sandal, Loafer,
Sandal }
```

Other important methods are *union*, *differ* and *intersect* which basically work the same as their equivalents in SQL. Consider the following:



```
Integer set X, set Y, set result;
X = Bag{1,2,3};
Y = Bag{3,4,5};
result = X.union(Y);
result.print();
end;
```



```
Bag{ 1,2,3,4,5 } (all in X added to all in Y)
```



```
Integer set X, set Y, set result;
X = Bag{1,2,3};
Y = Bag{3,4,5};
result = X.differ(Y);
result.print();
end;
```



```
Bag{ 1,2 } (those in X minus those in Y)
```



```
Integer set X, set Y, set result;
X = Bag{1,2,3};
Y = Bag{3,4,5};
result = X.intersect(Y);
result.print();
end;
```



```
Bag{ 3 } (both in X and in Y)
```



It is useful to print additional text in your queries. This can be done by defining a string variable and adding text to this. Then use the *print* method to display the text:

```
...
String textOutput;
textOutput = "This is a message";
textOutput.print();
...
```



Do not use undeclared variables or collections! If you fail to provide the instance, you cannot use the specified handle. Consider the following:

```

Integer set is1, set is2, set is3, set is4, set is5;
is1 = Bag{1,2,3,4};
is2 = Bag{1,3,5,7};
is3 = Bag{};
is4 = Bag{};
is3 = is1.union(is2);
is3.print();
is3 = is1.intersect(is2);
is3.print();
is4.directAdd(5);
is4.directAdd(81);
is4.print();
is4 = is4.add(73);
is4.directRemove(81);
is4.print();
is5.print();
is5.directAdd(5);
is5.print();

```

This would result in the following output:

```

Bag{ 4,2,1,3,5,7 }
Bag{ 1,3 }
Bag{ 5,81 }
Bag{ 5,73 }
NIL
NIL

```

This happens because the undeclared collection *is5* cannot be used since it has not been instantiated through the `= Bag{}` declaration. Collections or attributes that are NIL cannot be used! Also always use the method *directAdd* instead of the add method to put additional singletons in a collection.

Of course it is also possible to use ordinary program language constructs like *IF-THEN-ELSE* and *WHILE* in ODQL. The following example checks if any of the bags contains the integer value 2 and print out the result.



```

Integer set is1, set is2;
is1 = Set{1,2,3,4};
is2 = Set{1,3,5,7};
if (is1.hasElement(2)) {
    is1.print();
};
if (is2.hasElement(2)) {
    is2.print();
};

```



```

Bag{ 1,2,3,4 }

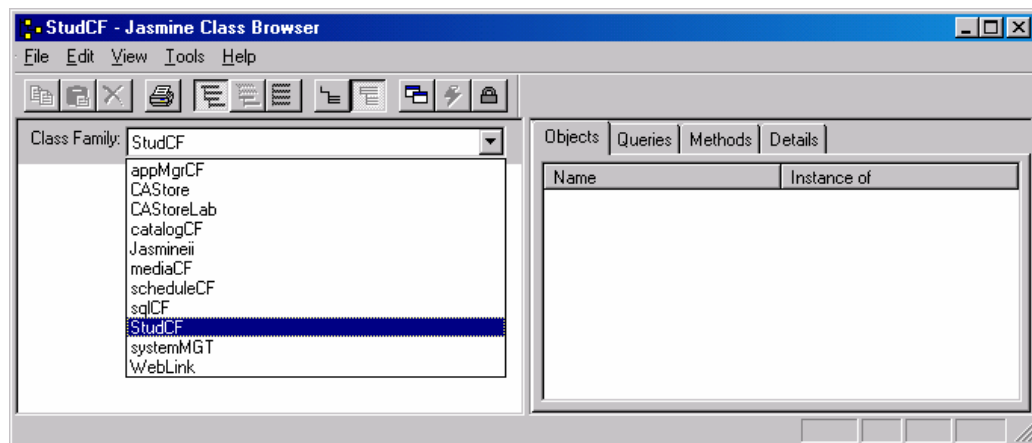
```

## 5 Modifying a Jasmine Database

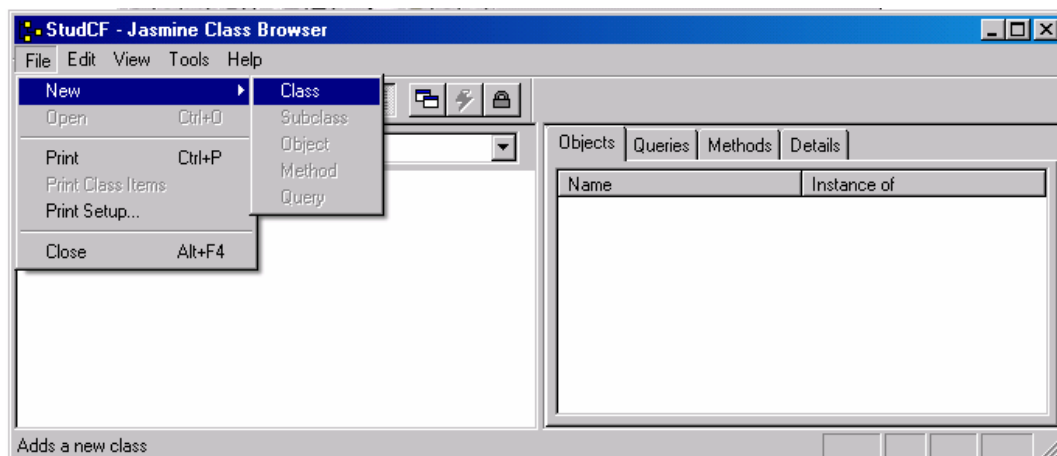
Up to now we have made no modifications to the Jasmine database but in this chapter we will see how to create new classes in a database (or class family), how to add attributes to these classes, how to create instances of classes (objects) and how to fill in values to their attributes.

### 5.1 Creating Classes

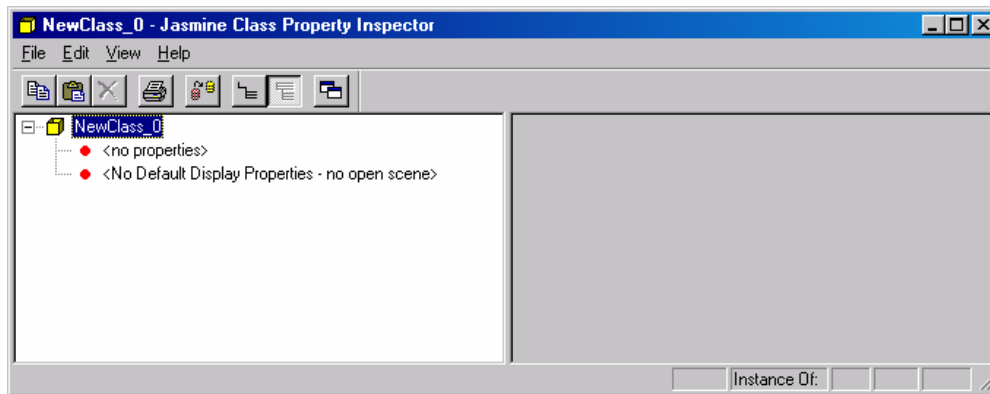
For the following examples we will use an already created and empty class family: *StudCF*.



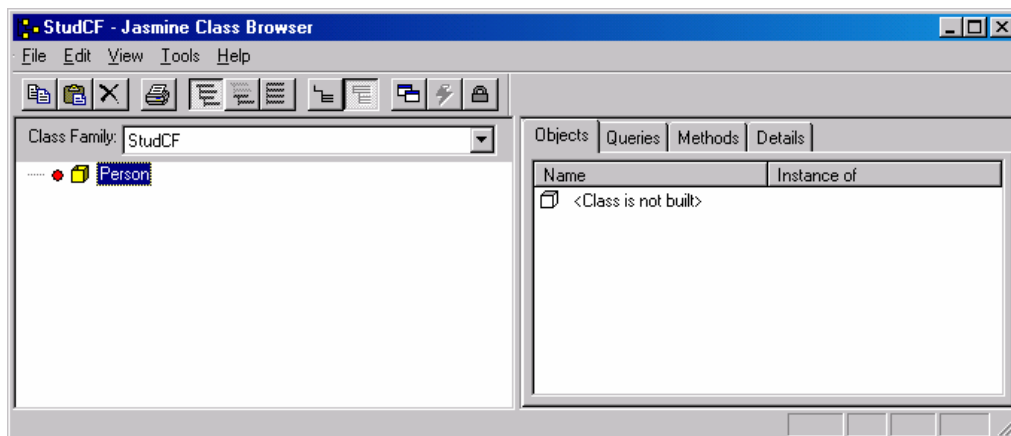
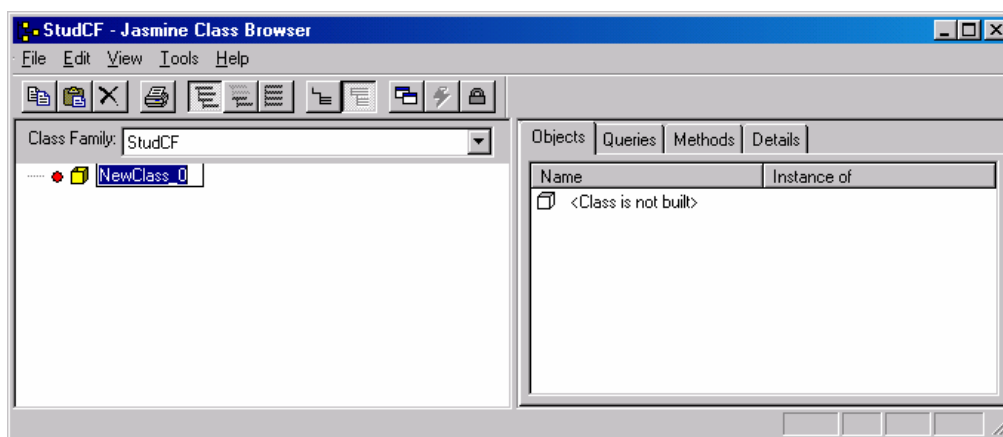
We will add classes representing people and the pets they have. Go to the menu **File » New » Class**



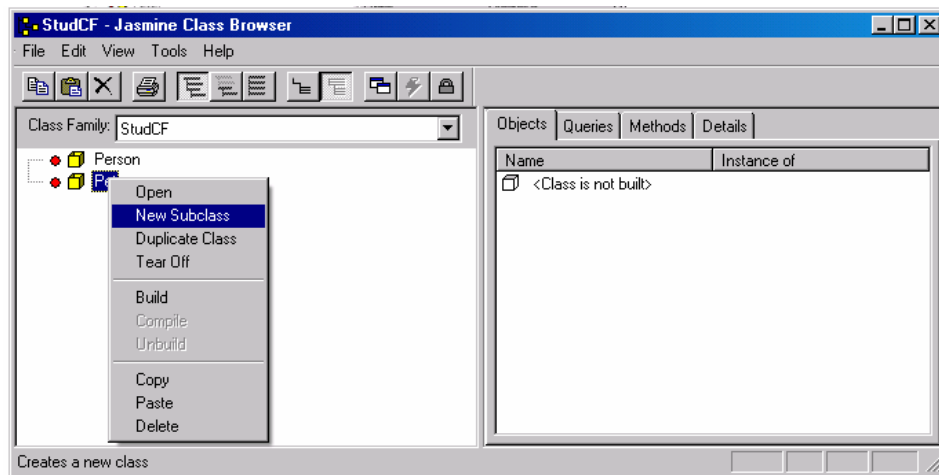
A new window will appear showing you the newly created class:



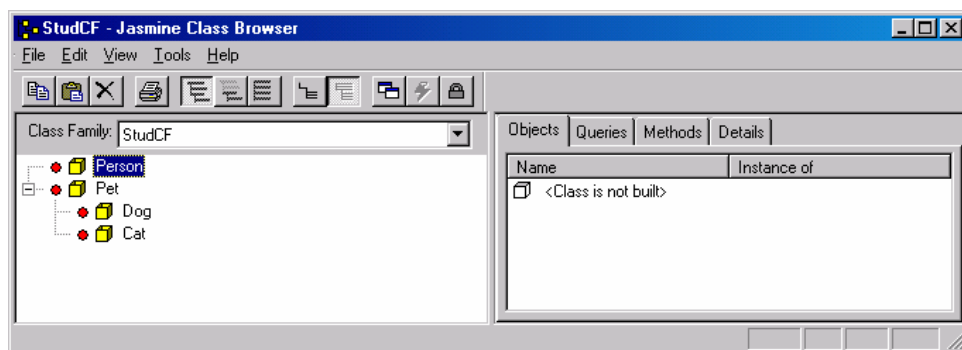
Close this window and click once on the name of the class to change it from "NewClass\_0" to "Person":



We proceed in the same way to create a new class called "Pet" and two subclasses of it called "Dog" and "Cat". To create a subclass, right-click on the name of the class and choose "New subclass" and proceed in the same way as indicated before.

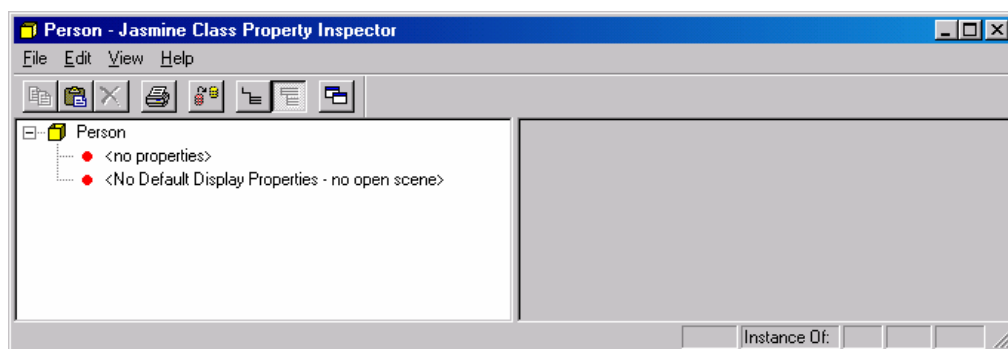


We will then end up with the following three classes:

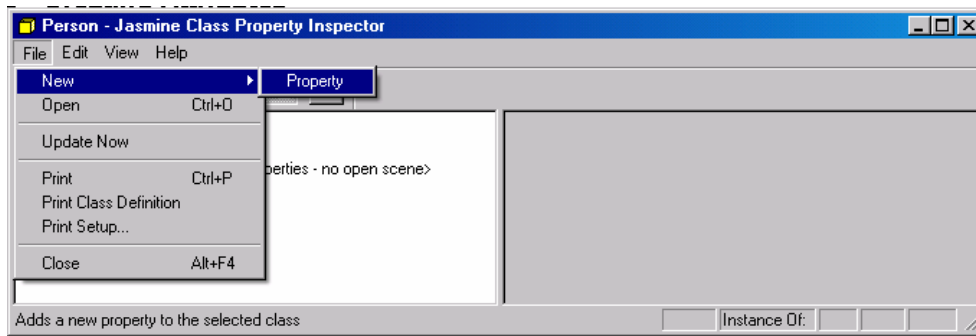


## 5.2 Creating Attributes

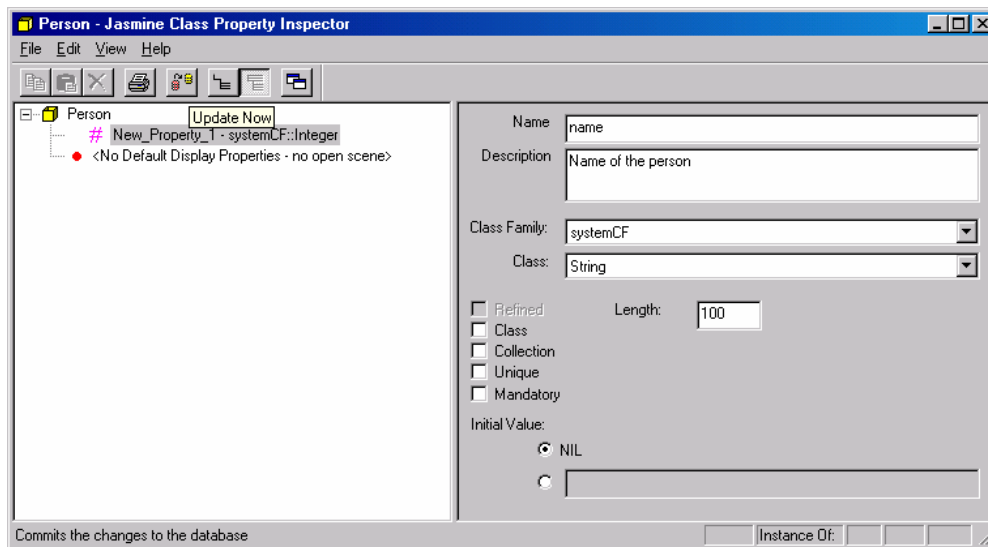
To add attributes to the classes we have already created we proceed in a similar way as before. So far we have used the *Jasmine Class Browser* to create the classes and now we will use the *Jasmine Class Property Inspector* to create attributes. We get to it by double-clicking in the class name we want to modify:



Go to the menu **File » New » Property**

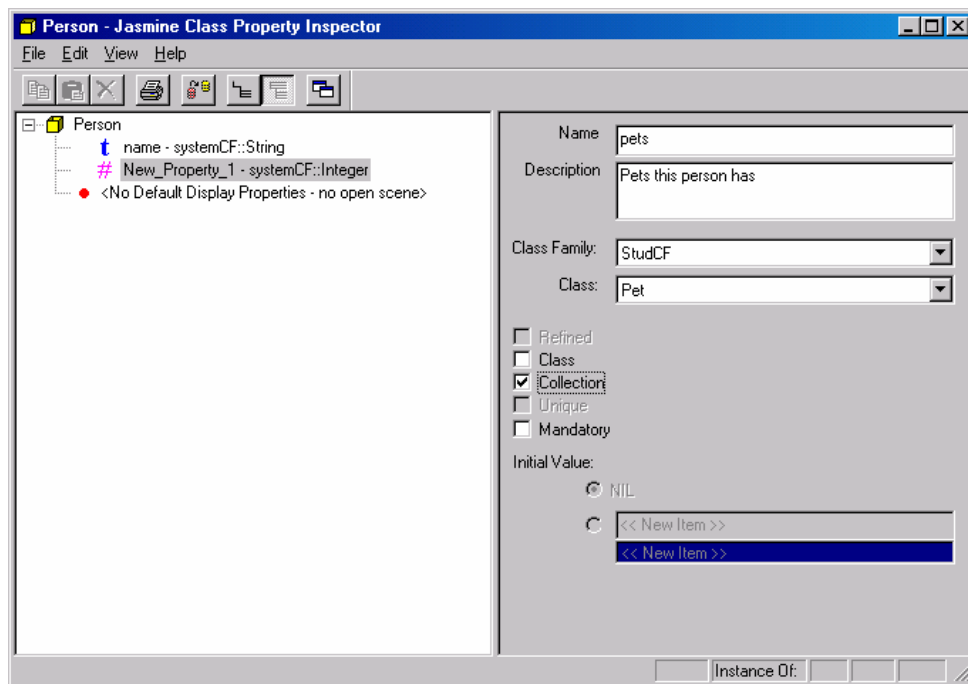


We will add the "name" attribute (or property in Jasmine's terminology). Notice that the type of the attribute will be "String" and that we have chosen a length of 100 characters.

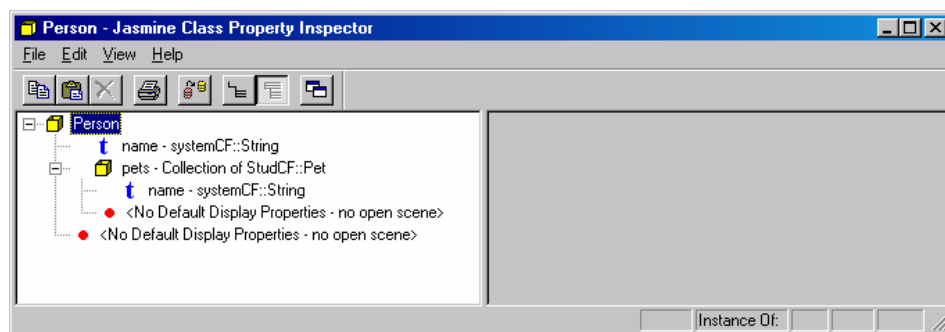


After you have filled in the values click on the toolbar button "Update Now" and the new attribute will be created. You should now proceed in the same way to add a "name" attribute to the class "Pet".

Once you have done this we can proceed to add a new attribute to *Person* that will contain the pets this person has. This time we will choose "*StudCF*" as the class family and "*Pet*" for the class of this attribute. As a person can have more than one pet this will be a multi-valued attribute and thus we will use a collection.

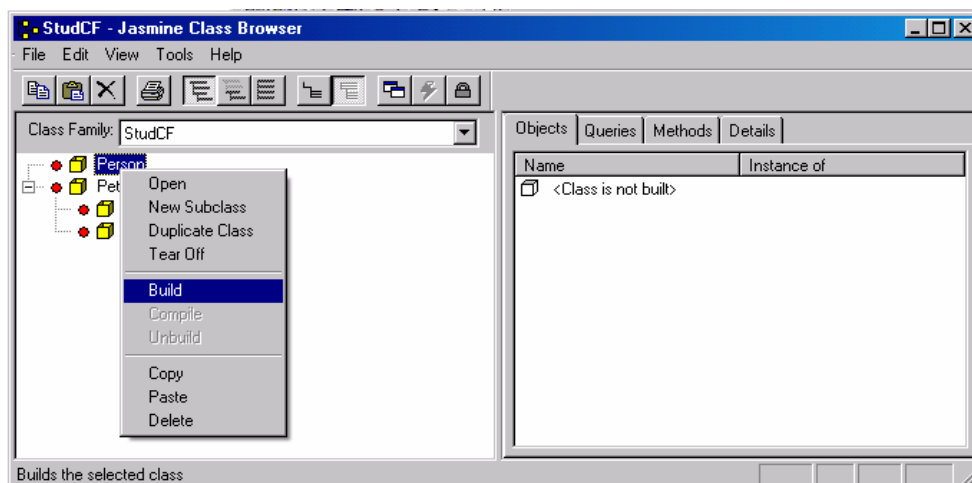


We finally obtain the following:

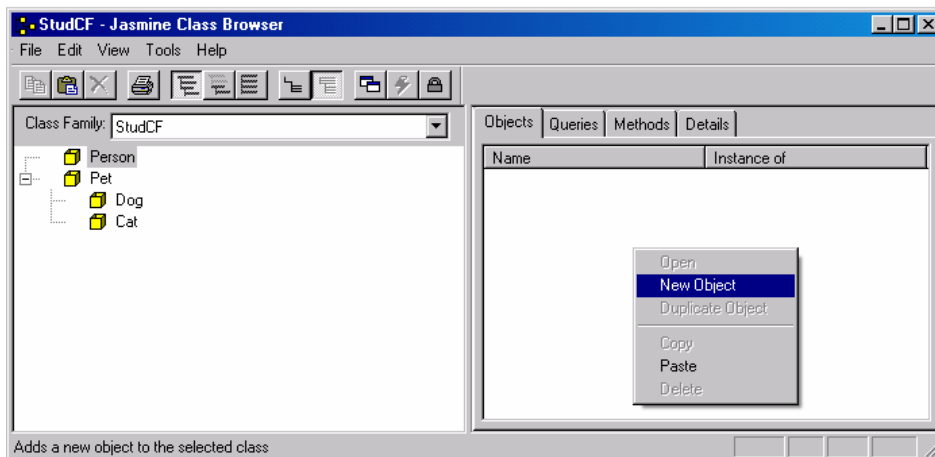


### 5.3 Creating Instances of a Class and Adding Values to Attributes

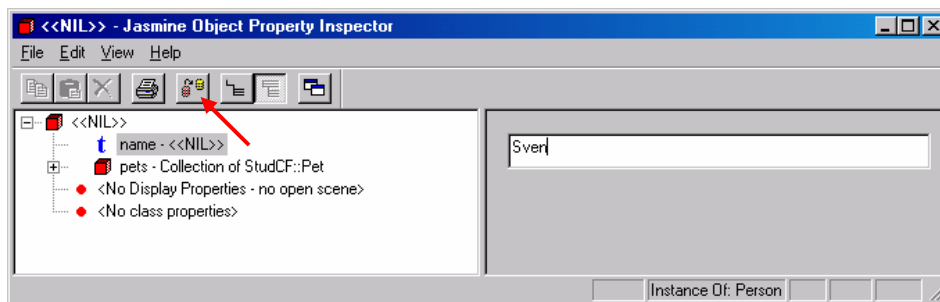
The red dot that appears next to the class name indicates that the class is not yet built and therefore we cannot create instances of it. Once we have created the needed classes and added the attributes, we need to build them.



After we have built all the other classes we can create instances (also called objects) of them. Select the class, right-click on the panel called "*Objects*" on the right hand side of the window and choose "*New object*".

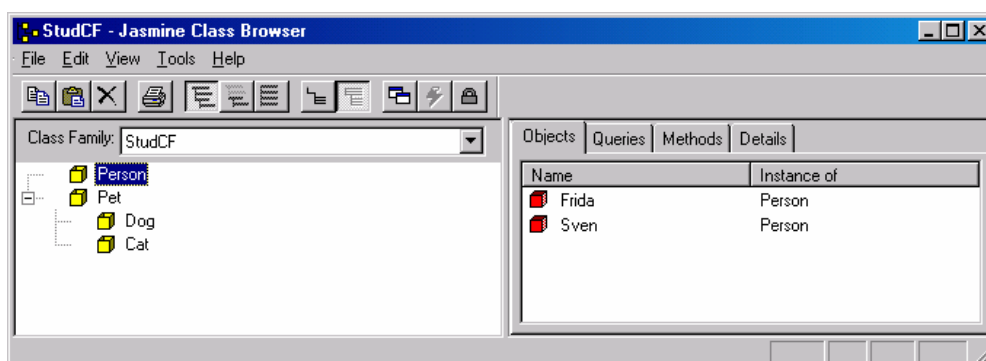


A new Jasmine Property Inspector window will open and you can fill in the values.



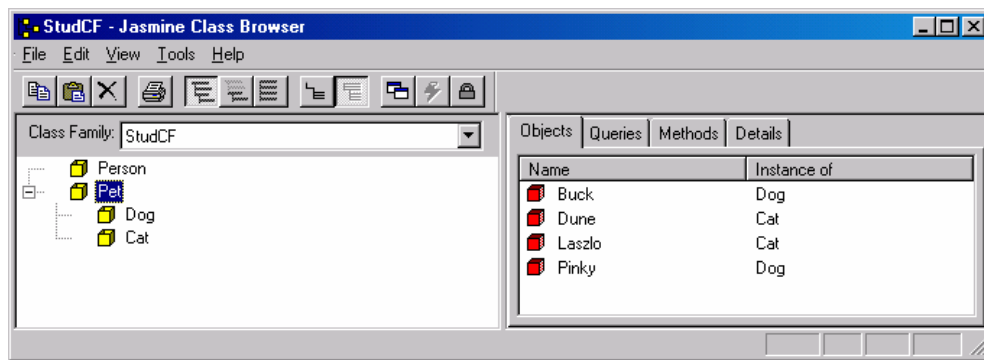
Click on the toolbar icon "*Update Now*" to commit the changes. We have now created an object of the class *Person* and have modified its "*name*" attribute. You should proceed in the same manner and create several objects for the other classes before we proceed to add values to the "*pets*" attribute of the class *Person*.

We have created two objects of the class *Person*

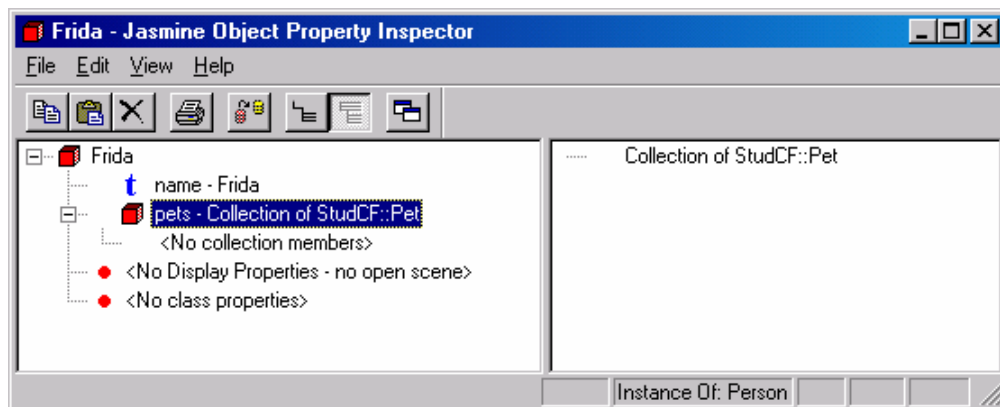


and four objects of the classes *Dog* and *Cat*

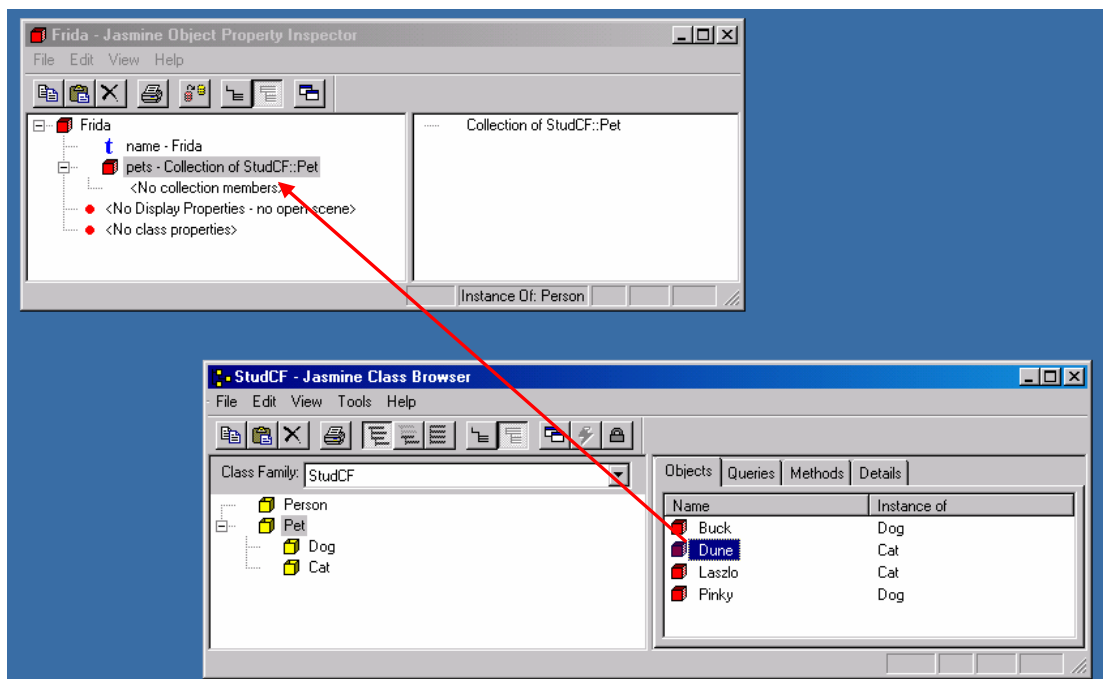




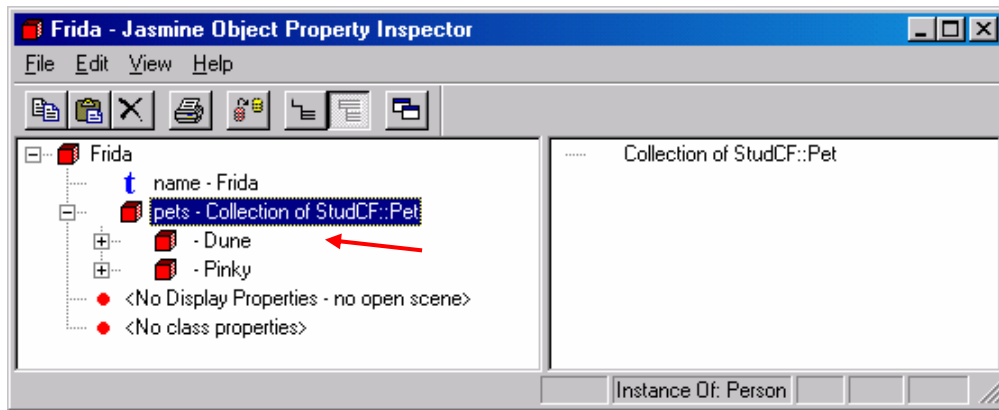
Now we will add values to the property "pets" of the class Person. Supposing Frida has one cat called Dune and one dog called Pinky, we will proceed as follows: double-click on the Frida object so a new *Jasmine Object Property Inspector* opens:



Now you can drag all the objects from the *Jasmine Class Browser* to the "pets" property in Frida:



After doing this we get the values we wanted:



## 5.4 Data Definition through ODQL

Everything that we have done in sections 5.1 to 5.3 can, of course, also be achieved through ODQL. To define new classes we use ODQL commands such as **defineClass** and **buildClass**. To add properties to a class we use methods such as **addProperty**. And to create new objects (instances of a class) we use the method **new()** and then assign values to the new object's attributes. You can find an explanation of these commands in the "*Jasmine ODB Database Developer Reference*". Go to the Windows menu **Start » Programs » Databases » Jasmine ii » Documentation » Database Developer's Reference**.

The scripts for creating the database of the previous sections are available here:

<\\db-srv-1\StudentCourseMaterial\IS4 spring 2006\Jasmine\>.

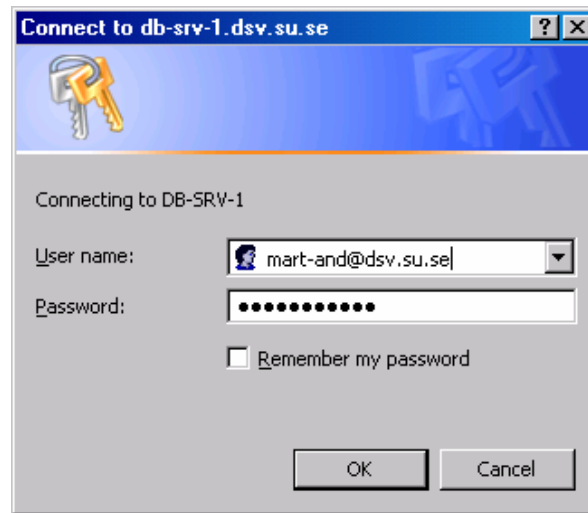
We provide you with a script to create the classes and their attributes (**CreateDB.txt**), another for creating instances (**PopulateDB.txt**), another for deleting the classes (**DropDB.txt**) and another to show the data (**ShowDB.txt**).

## 6 Creating a Copy of the CAStore Database

To solve one of the assignments you will need to modify the CAStore Jasmine database. As something can go wrong while modifying the database, we have created a script that will make a copy of the CAStore database. You can then **work on the copy** of the CAStore database and if something goes wrong then you can delete it and create a new copy. There are two scripts: *create\_db.bat* and *delete\_db.bat*.

These scripts are available at the DB-SRV-1 server. The files can be accessed at: <\\Db-srv-1\StudentCourseMaterial\IS4 spring 2006\Jasmine>

When you try to access the server a login prompt will appear. Type your personal DSV username (not db2admin), **together with the domain name** (@dsv.su.se), and your user password (for Windows) at the login prompt, as shown in the following example:



Click OK and you should now have access to the server and be able to copy the scripts to your disk. Copy them to the D: drive.



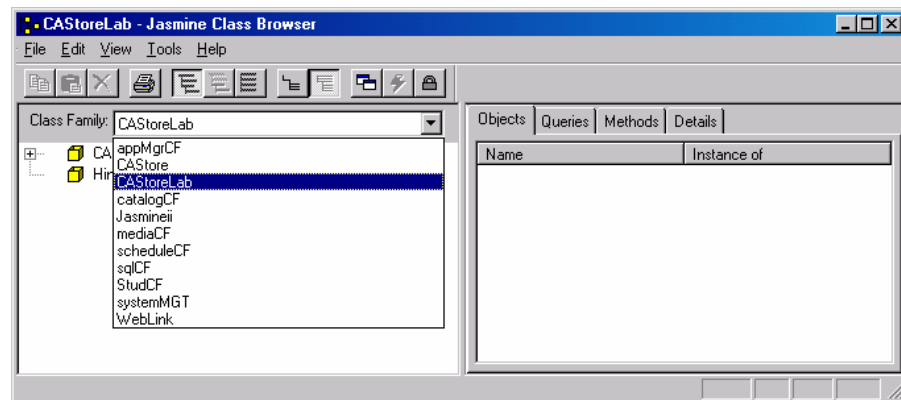
***Make sure you have closed all the windows of Jasmine Studio before executing any of these scripts!***



To create a copy of the CAStore database in order to solve the assignment open a *Command Prompt* window and run the create\_db batch file:

D:\>**create\_db**

Once the database has been created we can use it from *Jasmine Class Browser*:



If you make a mistake and want to delete the CAStoreLab database and start again from the beginning:



To delete the copy of the CAStoreLab do:

D:\>**delete\_db**



To create the copy of the CAStore again do:

D:\>**create\_db**

## 7 References

1. *Building a Jasmine Database – An Introduction*. Jasmine Conference, CA-World99. Peter Fallon. Castle Software Australia.  
<http://www.castlesoftware.com.au/Whitepapers/BuildingDatabase.pdf>