cbr:works 4

Introduction to the Case-Query-Language (Version 2)

Revision 2.0, 24 May 2000

tec:inno GmbH

Sauerwiesen 2
D-67661 Kaiserslautern
Tel: +49 (0) 6301 606 400
Fax: +49 (0) 6301 606 409

cbrworks@tecinno.com
support@tecinno.com
www.tecinno.com

Contents

1 Introduction to CQL2

This chapter introduces the syntax and use of CQL2, the case-query-language (version 2) of CBR-Works 4.

1.1 Summary

CQL2 is the case-query-language of the CBR-Works system (since version 4). It is the interface language between all the CBR-Works component systems and it also serves as the interface between the CBR-Works-Server and the external world (e.g. internet-clients). It is a standard for exchanging information.

CQL2 is an object-oriented language for storing descriptive models as well as cases (named *case library* or *casebase*) in ASCII-files. CQL2 is also used to exchange models and cases between related components using the network-transfer-protocol TCP/IP. CQL2 supports the representation of domain objects in a class hierarchy with inheritance and slots to describe the attributes of these domain objects.

CQL2 represents all the information related to a particular application domain in a common format (i.e. class descriptions, slots and their values, cases, etc.). The descriptive model defines the terms used to describe cases. It is stored in a file with a “.cdm“-extension. The case library (casebase) is stored in files with the extension “.cql“.

1.2 CQL2-Syntax by Example

The complete CQL2 syntax can be found in file *CQL-EBNF.pdf*. The following explains the syntax and semantics of CQL2 by the example given in the tutorial “Getting Started with CBR-Works - A Beginner’s Guide”, the *Used-Car-Domain* (see table 1-1):

Table 1-1: Car-Domain

Feature	Type	Range
Brand	Symbol	Audi, BMW, Fiat, Mercedes Benz, Opel, Seat, Skoda, VW
Mileage	Integer	1 to 500’000
Price	Integer	1 to 200’000
Colour	Symbol	black, blue, dark blue, green, dark green, red, dark red, white, yellow
Sunroof	Boolean	Yes, No
Hitch	Boolean	Yes, No
Catalytic Converter	Boolean	Yes, No
Seller	String	Free Text

The description of a domain in CQL2 is structured in few sections, as declared by the following EBNF-definition (EBNF=Extended Backus-Naur Form):

```
<descriptive model> ::=
   <basic model>
   {<measure definition>}*
   {<rule definition>}*

<basic model> ::=
   [<model properties>]
   {<type definition>}*
   {<value definition>}*
   {<slot definition>}*
   {<class definition>}+

<model properties> ::=
   <properties>
   version <string>
   serial <string> “.”
```

There are some sections in the entire descriptive model, model properties (optional), the basic model, measure definitions and rules. Model properties are e.g. the creator of a model and the version string of the model. The basic model contains all necessary data for client development, measures and rules are not needed for this intention.

The basic model consists of

- type-,
- value-,
- slot- and
- class definitions.

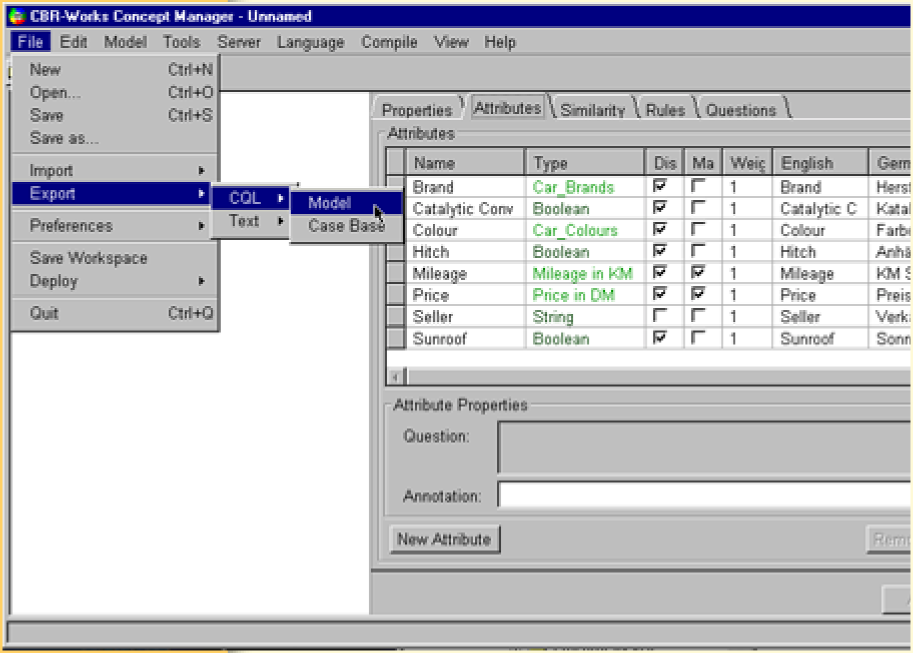
The plus-sign behind `class definition` indicates that there is at least one class required to define a valid model. The stars mean any iterations (incl. none) of the respective definition.

The sequence of this definitions is unalterable, the types have to be defined prior to slots, because slotdefinitions use typedefinitions.
Same argument to classdefinition, they use types and slots.

If you followed the tutorial “Getting Started with CBR-Works” then export the domain-model and its cases, i.e. Vectra1 and Vectra2 with the `File-menu` -> “`Export CQL..`” -> “`... Model`” or “`... Case Base`” (see figure 1-1).

Alternatively, you can copy&paste all of the following example-code into a textfile and import it to CBR-Works.

Figure 1-1: Export CQL



Attributes

Name	Type	Dis	Ma	Weig	English	Gern
Brand	Car_Brands	✓	☐	1	Brand	Herst
Catalytic Conv	Boolean	✓	☐	1	Catalytic C	Katal
Colour	Car_Colours	✓	☐	1	Colour	Farb
Hitch	Boolean	✓	☐	1	Hitch	Anhä
Mileage	Mileage in KM	✓	✓	1	Mileage	KM S
Price	Price in DM	✓	✓	1	Price	Preis
Seller	String	☐	☐	1	Seller	Verk
Sunroof	Boolean	✓	☐	1	Sunroof	Sonn

The exported-file (carDomain.cdm) should look like the following.

```
creator "Developer ()" date(1999 2 17) version
        "CBR-Works 4.0.8" serial "".
deftype "Price in DM" a_kind_of "Real"
   creator "Developer ()" date(1999 2 17) unit
         "DM"
   range [100.0 .. 200000.0]
   annotation "default" "".
```

comment "The price of a car is at least 100,- DM and a maximum of 200’000,- DM. Price is a subtype of Real so a car may be offered at a price of 9’999,99 DM, thus cheaper than 10’000,- DM. The first fact is given by the keyword *range* , the latter by *a_kind_of Real*. For a detailed description of this domain and the design decisions taken please refer to the tutorial “Getting Started with CBR-Works”.
Here we define a list of possible colours of the cars.".

```
deftype "Car_Colours" a_kind_of "Symbol"
   creator "Developer ()" date(1999 2 17)
   range ( "blue" "darkblue" "yellow" "green"
         "darkgreen" "red" "darkred" "black"
         "white" ).

deftype "Car_Brands" a_kind_of "Symbol"
   creator "Developer ()" date(1999 2 17)
   range ( "Audi" "Mercedes Benz" "VW" "Opel"
         "Skoda" "BMW" "Fiat" "Toyota"
         "Renault" ).
deftype "Mileage in KM" a_kind_of "Integer"
   creator "Developer ()" date(1999 2 17) unit
         "KM"
   range [0 .. 1000000]
   annotation "default" "".
```

comment "Definitions of the slot-attributes:
We define the attributes of our concept Used_Cars
which are:
Brand, Mileage, Price, Colour, Sunroof, Hitch,
Catalytic Converter and Seller
Each of them has a type, a weight and a print_name
`weight` is the weight, which you have specified at the
concepts-view of CBR-Works, default-value = 1
`print_name` is a set of names, one defaultname
and a name for each supported language, if you have
modelled with different languages”.

```
defslot "Brand" of "Used_Cars"
   type "Car_Brands"
   weight 1
   print_name "english" "Brand"
   print_name "german" "Hersteller"
   print_name "default" "Brand".
```

```
defslot "Mileage" of "Used_Cars"
   type "Mileage in KM"
   weight 1
   mandatory
   print_name "english" "Mileage"
   print_name "german" "KM Stand"
   print_name "default" "Mileage".

defslot "Price" of "Used_Cars"
   type "Price in DM"
   weight 1
   mandatory
   print_name "english" "Price"
   print_name "german" "Preis"
   print_name "default" "Price".

defslot "Colour" of "Used_Cars"
   type "Car_Colours"
   weight 1
   print_name "english" "Colour"
   print_name "german" "Farbe"
   print_name "default" "Colour".

defslot "Sunroof" of "Used_Cars"
   type "Boolean"
   weight 1
   print_name "english" "Sunroof"
   print_name "german" "Sonnendach"
   print_name "default" "Sunroof".

defslot "Hitch" of "Used_Cars"
   type "Boolean"
   weight 1
   print_name "english" "Hitch"
   print_name "german" "Anhängerkupplung"
   print_name "default" "Hitch".
```

```
defslot "Catalytic Converter" of "Used_Cars"
   type "Boolean"
   weight 1
   print_name "english" "Catalytic Converter"
   print_name "german" "Katalysator"
   print_name "default" "Catalytic Converter".

defslot "Seller" of "Used_Cars"
   type "String"
   weight 1
   not_discriminant
   print_name "english" "Seller"
   print_name "german" "Verkäufer"
   print_name "default" "Seller".
```

comment “Definition of all classes at CBR-Works:
(class is a synonym to the notion ’concept’)
`is_case_class` : If you have modelled more than one concept, you have to choose one as the case-concept, that means, you want to start a query for this concept. CBR-Works supports this at the concept-view by ’`Define as Case`’ at the context-menu.
`slots` : This is a collection of the slots which the class has.
`print_name` is a set of names, one defaultname and a name for each supported language, if you have filled the other language slots”.

```
defclass "Used_Cars" a_kind_of class
   creator "Developer ()" date(1999 2 17)
   is_case_class
   slots "Brand" "Mileage" "Price" "Colour"
         "Sunroof" "Hitch" "Catalytic Con-
         verter" "Seller"
   placement inplace
   print_name "default" "Used_Cars".
```

comment “ Please note, all user-defined identifiers of types and slots are in quotes (“<type identifier>“).

Now an example of measure-definition:
Car_Colours_Similarity
It is defined by a similarity-table”.

```
defmeasure "Car_Colours_Similarity" of_type
        "Car_Colours"
   creator "Developer ()" date(1999 2 17)
   mode "table"
   parameters "#(#(
        #(1.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8)
        #(0.1 1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3)
        #(0.2 0.9 1.0 0.2 0.1 0.2 0.3 0.4 0.5)
        #(0.3 0.8 0.2 1.0 0.6 0.7 0.8 0.9 0.8)
        #(0.4 0.7 0.1 0.6 1.0 0.7 0.6 0.5 0.4)
        #(0.5 0.6 0.2 0.7 0.7 1.0 0.3 0.2 0.1)
        #(0.6 0.5 0.3 0.8 0.6 0.3 1.0 0.2 0.3)
        #(0.7 0.4 0.4 0.9 0.5 0.2 0.2 1.0 0.4)
        #(0.8 0.3 0.5 0.8 0.4 0.1 0.3 0.4
        1.0))
        #symmetric)".
defmeasure "intra" of_class "Used_Cars"
   creator "TECINNO GmbH" date(1998 11 1)
   mode "sumation".
```

The casebase file carDomain.cql looks as follows:

```
add_case
   defcase 1 confirmed
   creator "Developer ()" date(1999 2 17)
   objects
   "Used_Cars" "id1"
   "Sunroof" : true,
   "Mileage" : 70400,
   "Seller" : "Touring Garage GmbH",
   "Brand" : "Opel",
   "Hitch" : true,
   "Colour" : "red",
   "Price" : 11900.0,
   "Catalytic Converter" : true
   print_name "default" "Vectra1".


add_case
   defcase 2 confirmed
   creator "Developer ()" date(1999 2 17)
   objects
   "Used_Cars" "id1"
   "Sunroof" : true,
   "Mileage" : 25000,
   "Seller" : "Opel Johannes",
   "Brand" : "Opel",
   "Hitch" : false,
   "Colour" : "blue",
   "Price" : 24450.0,
   "Catalytic Converter" : true
   print_name "default" "Vectra2".
```

`add_case` : This stands for “adding a new case”.
`defcase` : ’define a case’ ...
`objects` : ’with following objects...’

These are the basics of the description of a simple domain-definition in CQL. Now, the next chapter will show how to utilise and deploy CQL.

2 Utilising CQL2

This chapter gives two examples for utilising CQL2, “correcting mistypings” and “CQL2 as interface between CBR-Works and your applications”.

2.1 Correcting mistypings

Suppose you have defined your model and just put some cases into the casebase. Sometime you may notice a typing mistake in your model or you do not want name the producer “brand” any longer, instead you want it to be shown as “manufacturer”. CBR-Works does not support an user-friendly way to rename automatically a type in the casebase and modulate the case base. If you simply rename the slot “Brand” into “Manufacturer” then all entries in the cases are renamed, but the values of the Brand Manufacturer are lost. A “work-around” utilising CQL2 will do the job. This work-around renames all occurrences of Brand into Manufacturer. If you want only to change the print-name, you can do this with the CBR-Works4 application: Change only the printname “english” with your desired Manufacturer and change the language of you application.

Caution: Always make a backup of your model and casebase files before you change them manually.

Open the exported model and the casebase files in a text-editor. Now you can replace the string “Brand” with “Manufacturer”: change...

```
defslot "Brand" of "Used_Cars"
   type "Car_Brands"
   weight 1
   print_name "english" "Brand"
   print_name "german" "Hersteller"
   print_name "default" "Brand".
to
```

```
defslot "Manufacturer" of "Used_Cars"
   type "Car_Brands"
   weight 1
   print_name "english" "Manufacturer"
   print_name "german" "Hersteller"
   print_name "default" "Manufacturer".
```

and rename slot "Brand" to "Manufacturer" in the class (concept) Used_Cars

```
defclass "Used_Cars" a_kind_of class;
        is_case_class;
        slots "Brand" "Mileage" "Price" "Col-
        our" "Sunroof" "Hitch" "Catalytic Con-
        verter" "Seller";
        print_name "default" "Used_Cars".
to

defclass "Used_Cars" a_kind_of class;
        is_case_class;
        slots "Manufacturer" "Mileage" "Price"
        "Colour" "Sunroof" "Hitch" "Catalytic
        Converter" "Seller";
        print_name "default" "Used_Cars".
```

It is not necessary to rename the String “Car_Brands” in the slot-definition because this name will not be shown to the user, but if you do so, do not forget to rename “Car_Brands” into “Manufacturer_Enumeration” or alike.

```
deftype "Car_Brands" a_kind_of "Symbol"
   creator "Developer ()" date(1999 2 17)
   range ( "Audi" "Mercedes Benz" "VW" "Opel"
        "Skoda" "BMW" "Fiat" "Toyota"
        "Renault" ).
deftype "Mileage in KM" a_kind_of "Integer"
   creator "Developer ()" date(1999 2 17) unit
        "KM"
   range [0 .. 1000000]
   annotation "default" "".
```

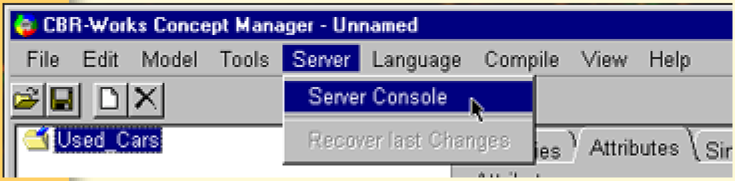
2.2 CQL2 as interface between CBR-Works and your applications

CBR-Works allow to connect to other applications via CQL2 and TCP/IP. CBR-Works then starts a Server. The easiest way to show this is by trying TELNET:

First you have to set up the server-process:

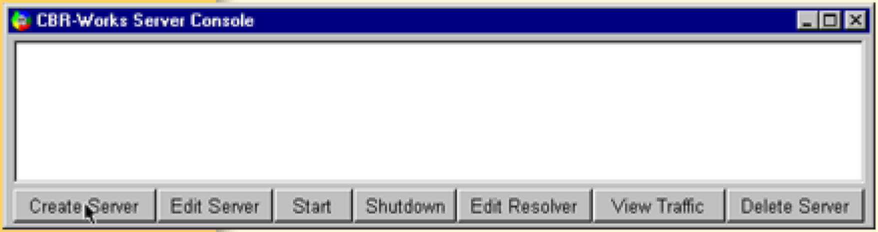
- Start your CBR-Works-Application
- Select “Server-Console” in the Server-menu of CBR-Works.

Figure 2-1: Menu: Server Console



- Create a CQL-Server with “Create Server” if there’s none running:

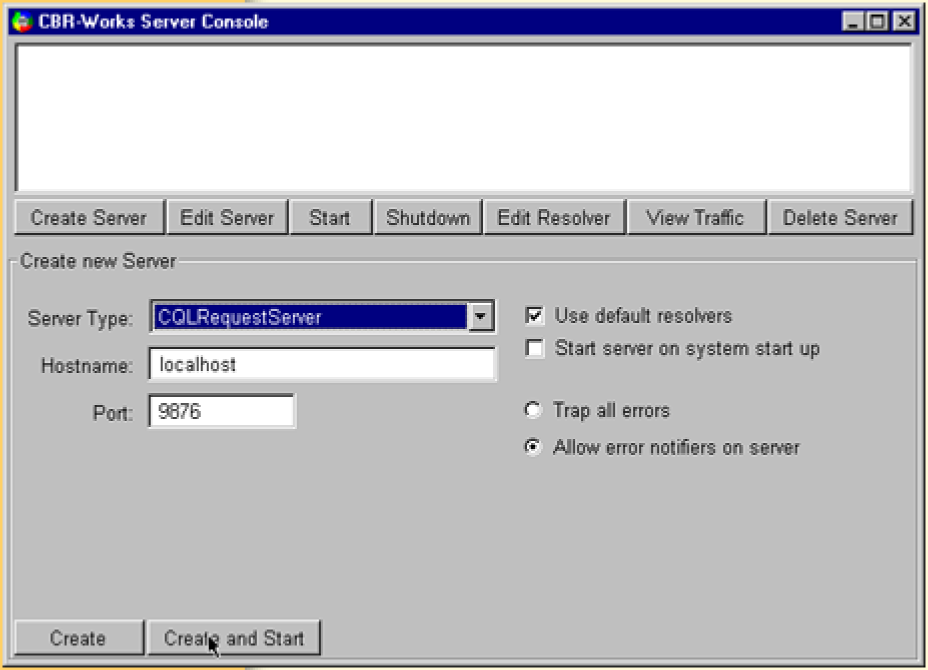
Figure 2-2: Create Server



- For “Server Type” select CQLRequestServer
- Hostname should be “localhost” if it should run on your own PC, the respective machineaddress else.

- Remember or note the displayed port-number for the later use in telnet.
- Select Trap all errors, if you do not want to get exceptions if you make some mistakes.
- Last: Select “Create and Start”

Figure 2-3: Server Console Entries



Server Type: CQLRequestServer
Hostname: localhost
Port: 9876

[✓] Use default resolvers
[ ] Start server on system start up
( ) Trap all errors
(●) Allow error notifiers on server

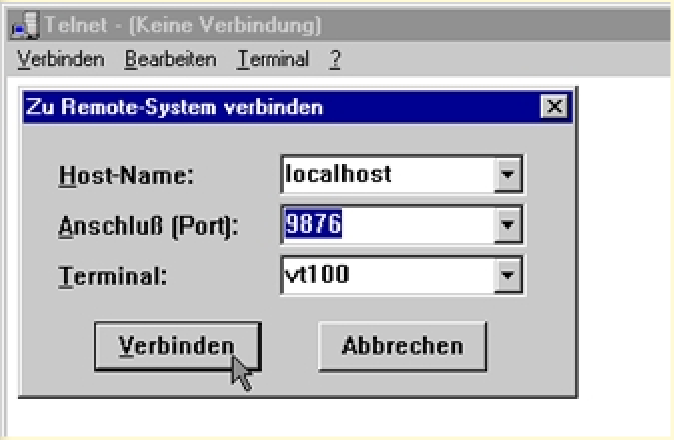
Now start telnet.

Establish a connection to your pc “localhost” or the machine where CBR-Works is running

- For Host-Name enter localhost or the remote-machine.

- The port has to be the port-number, which you chose during the server-setup, (default = 9876). Enter the appropriate number.
- select “connect” to establish your connection.

Figure 2-4: Telnet Client


Telnet - (Keine Verbindung)
Verbinden Bearbeiten Terminal ?
Zu Remote-System verbinden
Verbinden
Abbrechen

Host-Name: localhost
Anschluß (Port): 9876
Terminal: vt100

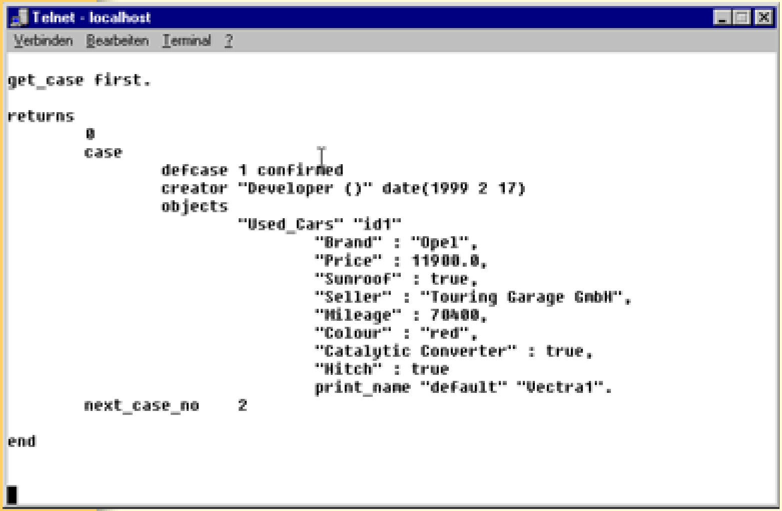
Do not forget to activate the local echo (`->Terminal->Settings->Local echo`), that you see what you are typing.

Use CQL2 to interact with the server, i.e. type

```
get_case first.
```

... and hit the ”ESC”-key to transmit the command to the server! You will get the first case of the Case-Base. The output should look like the one below.

Figure 2-5: Telnet Output



```
get_case first.

returns
        0
        case
                defcase 1 confirmed
                creator "Developer ()" date(1999 2 17)
                objects
                        "Used_Cars" "id1"
                                "Brand" : "Opel",
                                "Price" : 11900.0,
                                "Sunroof" : true,
                                "Seller" : "Touring Garage GmbH",
                                "Mileage" : 70400,
                                "Colour" : "red",
                                "Catalytic Converter" : true,
                                "Hitch" : true
                                print_name "default" "Vectra1".
        next_case_no    2

end
```

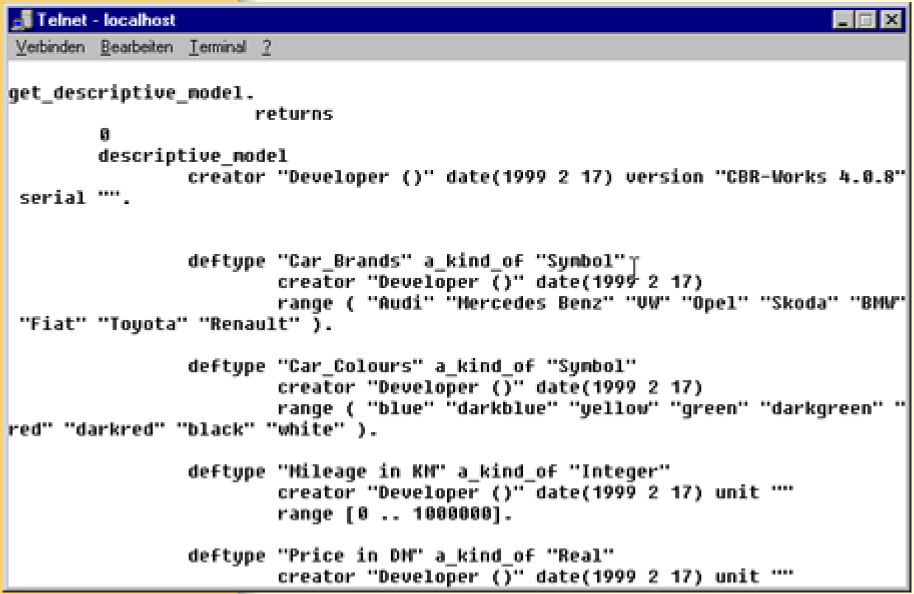
Directly above the “end” there’s the number of the next case (*next_case_no*), with this number you can build the next *get_case*-command and so on.

Or try

- `get_descriptive_model. <ESC>`

... and you get the whole model of your domain.

Figure 2-6: Get Descriptive Model



```
get_descriptive_model.
                                returns
            0
            descriptive_model
                        creator "Developer ()" date(1999 2 17) version "CBR-Works 4.0.8"
 serial "".


                        deftype "Car_Brands" a_kind_of "Symbol"
                                    creator "Developer ()" date(1999 2 17)
                                    range ( "Audi" "Mercedes Benz" "VW" "Opel" "Skoda" "BMW"
 "Fiat" "Toyota" "Renault" ).

                        deftype "Car_Colours" a_kind_of "Symbol"
                                    creator "Developer ()" date(1999 2 17)
                                    range ( "blue" "darkblue" "yellow" "green" "darkgreen" "
red" "darkred" "black" "white" ).

                        deftype "Mileage in KM" a_kind_of "Integer"
                                    creator "Developer ()" date(1999 2 17) unit ""
                                    range [0 .. 1000000].

                        deftype "Price in DM" a_kind_of "Real"
                                    creator "Developer ()" date(1999 2 17) unit ""
```

Note:

- Some of telnet-applications don’t echo the user-input. Because CBR-Works does not too, you should activate the local echo so that you can read what you are typing.
- You can type more than one line using <CR>, this will not transmit your input-stream, only the ESC-key does.

Each query that follows the syntax of the CQL2 query statement is a valid one:

```
<query> ::=
   send_query <no. of cases to retrieve>
          <threshold> [complete_query]
   [case_class <class_identifier>
          [including_subclasses]]
   [case_references_only | adapt_cases]
   [questions <no. of paths to return> [<no.
          of relevant cases for question com-
          putation>]]
   objects { <query object> \ ";" } "."
```

The keywords in [...] are optional. The words enclosed with <...> are references to another definitions.

Searching for the two vectra of the used car application can be achieved by typing... (followed by the ESC-key).

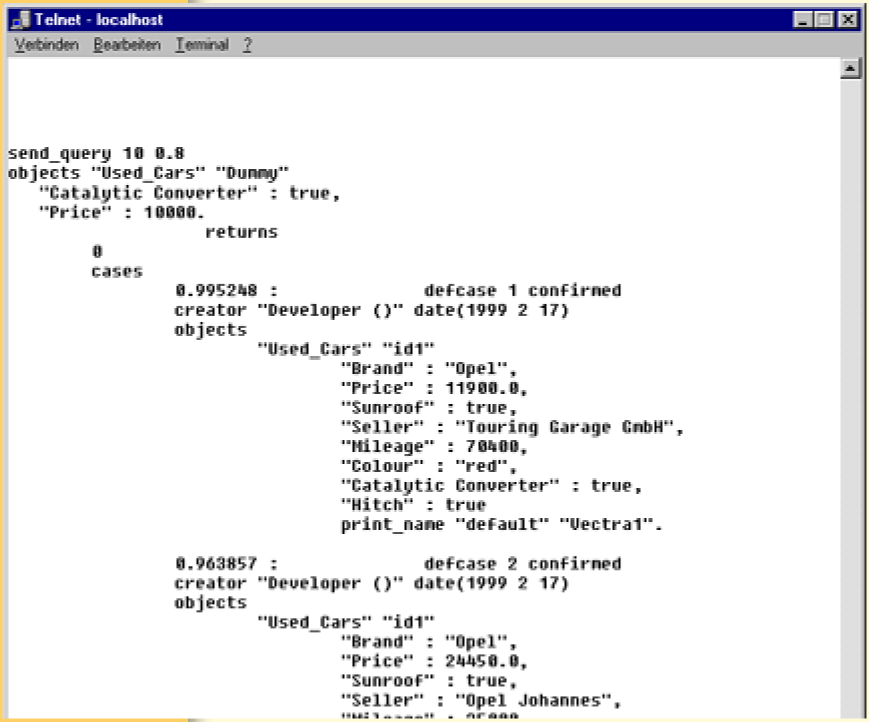
```
send_query 10 0.8
objects "Used_Cars" "Dummy"
  "Catalytic Converter" : true,
  "Price" : 10000.
```

This will retrieve both cars of the tutorial domain
(see figure 2-7).

Only the number of cases to retrieve, the treshold and the object definition is mandatory. The object definition consists of a class identifier (our concept ’Used_Cars’) and a local object idenitifier. The latter one is needed for sophisticated models, e.g. your model defines concepts as slots of another concept. In such a case it is necessary to firstly describe all the parts of a complex query and define a local idenitifier for each one, so you can refer them in the complex concept query.

Above you could replace the “Dummy” identifier with ““ because, we do not have any complex concept.

Figure 2-7: Query Example 1

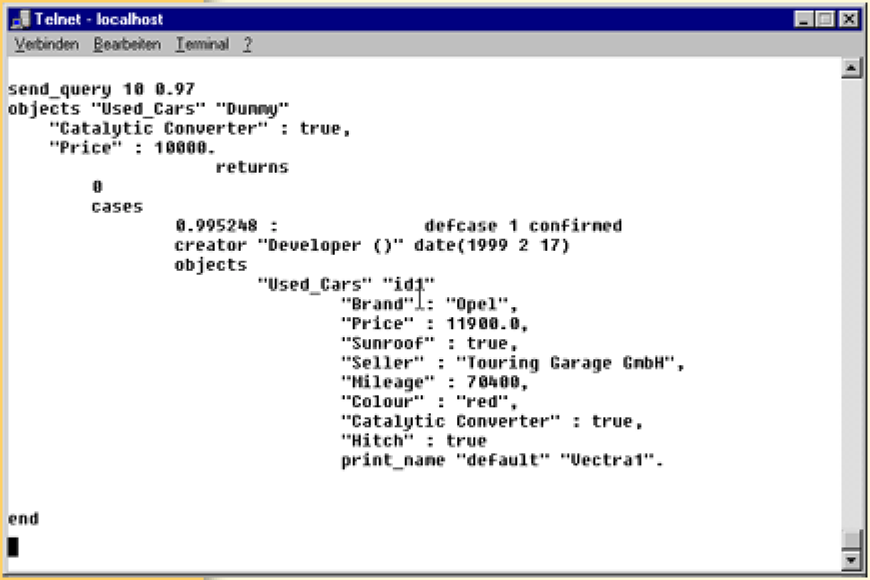


```
send_query 10 0.8
objects "Used_Cars" "Dummy"
   "Catalytic Converter" : true,
   "Price" : 10000.
                   returns
        0
        cases
                0.995248 :                defcase 1 confirmed
                creator "Developer ()" date(1999 2 17)
                objects
                        "Used_Cars" "id1"
                                "Brand" : "Opel",
                                "Price" : 11900.0,
                                "Sunroof" : true,
                                "Seller" : "Touring Garage GmbH",
                                "Mileage" : 70400,
                                "Colour" : "red",
                                "Catalytic Converter" : true,
                                "Hitch" : true
                                print_name "default" "Vectra1".

                0.963857 :                defcase 2 confirmed
                creator "Developer ()" date(1999 2 17)
                objects
                        "Used_Cars" "id1"
                                "Brand" : "Opel",
                                "Price" : 24450.0,
                                "Sunroof" : true,
                                "Seller" : "Opel Johannes",
```

Now restrict the threshold-value to 0.97 and you will receive only one vectra, the one which has a price closer to 10000 DM.

Figure 2-8: Query Example 2



```
send_query 10 0.97
objects "Used_Cars" "Dummy"
    "Catalytic Converter" : true,
    "Price" : 10000.
                    returns
        0
        cases
                0.995248 :               defcase 1 confirmed
                creator "Developer ()" date(1999 2 17)
                objects
                        "Used_Cars" "id1"
                                "Brand" : "Opel",
                                "Price" : 11900.0,
                                "Sunroof" : true,
                                "Seller" : "Touring Garage GmbH",
                                "Mileage" : 70400,
                                "Colour" : "red",
                                "Catalytic Converter" : true,
                                "Hitch" : true
                                print_name "default" "Vectra1".


end
```