Visual Classworks DDE Interface

1Terminology

Prior to leaping into the wonders of extracting design information from the tool via DDE it is important to get an understanding of a few DDE terms. A DDE communication takes place between two applications. One must take on a client role and the other a server role. Typically the DDE server is the one supplying the data and the client is the one consuming the data. In this case the tool acts as the server because it is providing the data and Word or some other DDE capable application is the DDE client.

Microsoft Word has a basic language built into it called WordBasic. Macros (functions) can be written in WordBasic to control and create documents. WordBasic supplies a set of DDE functions including DDEInitiate, DDETerminate, DDERequest and DDEExecute. Other Windows word processors may also supply a set of DDE functions. You could even write your own DDE client in C or C++ to communicate with the tool.

An example Word macro that communicates with the tool via DDE is included in the oodesign.dot template. The macro name is ExtractDesign. It can be opened by selecting Tools|Macros and double clicking on the ExtractDesign entry.

2Example Word Macros

The Microsoft Word document template oodesign.dot contains two macros: ExtractDesign and ProgressReport which are simple examples showing how to extract information from the tool. To use these macros you should copy oodesign.dot to your Word for Windows template directory. This is typically C:\MSOFFICE\WINWORD\TEMPLATE.

Open up the Visual Classworks with your design loaded and then run one of the macros using Tools|Macro from within Word. The macros will create a new document and fill it with information from your currently loaded model.

3Design

Before one can extract design information it is important to have a knowledge of what information is available and how that information is stored.

3.1Model

The model used in the tool consists of two fundamental architectural entities: objects and containers: Objects contain data and can have certain commands executed on them. Containers are collections of objects. Objects themselves can have containers which hold other objects. The object types in the model are listed below:

Table 1. Object Types and the names used to refer to them in DDE.

Model	model	The currently open design in the tool.
Category	category	A logical grouping of related classes - only one per model at this stage
Class	class	A class in the category
Base Class	baseClass	A base class of the current class
Data	data	A data member of the current class
Method	method	A method of the current class
Relation	relation	***An relationship (not inheritance) with another class.

***Relations are not implemented in Version 4.0 Beta.

A client never sees containers - they work with iterators which are internal structures that can navigate through containers. Each iterator maintains an internal cursor which points to the current object in its collection. Commands are provided to move the cursor to the next or previous object in a collection. Each time an object is navigated to any iterators on containers that objects of that type have are destroyed. The set up iterator command (*itObjectName - see below*) must be called for the new object before iterating on one of its containers.

While ever a model file is loaded in the tool there is always a valid model. In this version of the tool only one Category object is supported.

An understanding of the containment hierarchy (how objects contain one another) is prerequisite and extremely helpful in understanding how a client would navigate and access data in a model.

```
Model has_many Category has_many Class has_many Base Class
                                       has_many Data
                                       has_many Method
                                       has_many Relation
```

3.2Iterating Example

Let's say that the Category iterator is pointing to a valid category object (there is only one category at this stage in all designs). The client can execute itClass which will create an iterator on the classes in the category. The interator's cursor will point to the first class in the container or NULL if there is no classes in the category. If there is no class in the category then DDE_FNOTPROCESSED will be returned from the DDEExecute(itClass) command. A client must use this return value to determine if there is a valid object from which data can be extracted. In WordBasic the return value can not be

tested directly, rather an error will be generated if DDE_FNOTPROCESSED so the code must be enclosed in an On Error construct like this:

```
On Error Goto ErrorHandler
DDEExecute(channel, "category itClass")
‘ do interesting stuff with current class here
...
Goto Done
ErrorHandler:
' Do nothing
Done:
‘ Move onto next bit
```

Once itClass has been executed a user can get data from the current class by calling a set of access functions including: class name, class description.

To iterate to the next class a client executes `class next` or to the previous class by executing `class previous`. Both of these functions will return DDE_FACK or DDE_FNOTPROCESSED depending on whether the iterator is left pointing to a valid class object.

Whenever an iterator moves to another object all of the iterators that were created while the cursor was at the original object are destroyed. A client must issue another itXXX command to iterate through a container in the new object. Iterators above the original object in the containment hierarchy remain intact.

For example: the Category’s class interator remains intact when a class’ data iterator moves to the next data member. When the class iterator moves to the next class any BaseClass, Data, Method or Relation iterators are destroyed.

3.3Data extraction and Executing Commands

It is important to note the difference between extracting data and executing commands. Data extraction involves getting data from the server and uses the DDERequest (in WordBasic) or equivalent function . The execution of commands performs some internal action in the DDE server (the tool) and does not return anything other than a succeed or fail status. Command execution uses the DDEExecute (in WordBasic) or equivalent function.

3.3.1Extracting data from the current object

To access data of the current object in an iterator perform a DDERequest with the following argument format:

```
ObjectType DataName
```

For example to get a class’ Name use:

```
DDERequest(“class name”)
```

3.3.2Executing a command on the current object

To execute a command on an object use the following argument form:

```
ObjectType Command
```

For example to set up an iterator for the data members in a class:

```
DDEExecute(“class itData”)
```

The following lists the data and commands associated with each of the object types that can exist in the design. Note that for certain object types there are no commands.

Table 2. Model Data and Commands

name	Name of this model (Set in Options|Project)
designer	Name of the designer of this model. (Set in Options|Project)

itCategory Setup an iterator on the categories in this model.

Table 3. Category Data and Commands

copyright	Copyright notice for this category (Set in Options| Project)

itClass Setup an iterator on the classes in this category.
next Iterate to the next category in the model.

Table 4. Class Data and Commands

name	Name of the class
file	Name of file containing class minus extension
hext	Header file extension
cext	Source file extension
isAbstract	“t” for true or “f” for false
isTemplate	“t” for true or “f” for false
isLib	“t” if library “f” if not.
tempArgs	Template arguments
storage	“persistent” or “transient”
phase	2 = analysis, 4 = design, 6 = implementation, 8 = testing, 10 = complete
description	Description of the class

next	Iterate to the next class in the category - fails if no more classes
itBaseClass	Setup an iterator on the base classes of this class
itDataMember	Setup an iterator on the data members of this class

itMethod	Setup an iterator on the methods (functions) of this class
itRelation	Setup an iterator on the relations of this class

Table 5. Base Class Data and Commands

access	returns “public” or “protected”
virtual	returns “virtual” if it is a virtual base class or empty string “” if not.

Table 6. Data member Data and Commands

name	Name of the data member
type	Type of the data member
access	“public”, “protected” or “private”
phase	2 = analysis, 4 = design, 6 = implementation, 8 = testing, 10 = complete
description	Description of the data member

Table 7. Method Data and Commands

name	Name of the method
type	Type of the method
parameters	Parameters of the method
access	“public”, “protected” or “private”
dec	declaration as it appears in the code
phase	2 = analysis, 4 = design, 6 = implementation, 8 = testing, 10 = complete
description	Description of the method

Relations are not available in the 4.0 beta

Table 8. Relation Data and Commands

thisCard	Cardinality of this class with respect to the other: “one” or “many”
thisRole	Role name of this class with respect to the other
thatClass	Other class involved in this relationship
thatCard	Cardinality of the other class with respect to the this one: “one” or “many”
thatRole	Role name of the other class with respect to this