Class Methods

April 2004
John Green, Joanju Limited

Revision 00

Abstract

This document discusses some of the issues and a potential solution for simulating object method calls within the 4GL.

Challenges

- given multiple objects of the same class, how do we tell *getMyValue()* which object we want *myValue* from? This is normally dealt with by using function declarations with *IN proc-handle*, but...
- given multiple classes with different signatures for functions all named *getMyValue*, how do we avoid name collisions?
- how can the IDE provide a context sensitive (all methods for just one class) code completion?

Proposal

- Within a "class", use a FUNCTION definition whenever there is a single return value from a method.
- Otherwise, use a PROCEDURE definition for the method.
- We will use Proparse within the IDE to generate a header file for the class, which will contain function forward declarations similar to the following:
  function {1}%getValue returns logical map to getValue in {1}.

Usage Example

```
def var myObject as handle.
run ThingOne.p persistent set myObject.
{ThingOne.h myObject}
display myObject%getValue().
```

Benefits

By using a combination of a name prefix along with the MAP TO feature, we have taken care of all three of the challenges listed in the "Challenges" section, above. The code

completion within the IDE becomes feasible because the IDE would have a list of all functions available, sorted by name. By the time you typed in *myObject*, the list of completions would be fairly well narrowed down.

About the "%" in the Name Prefix

No, it's not necessary, and no, it does not matter if you use it, use nothing, or use something else. It's a valid name character, and I think it makes a reasonable name separator, since '.' and ':' are not available. We will make it configurable on a per-project basis within the IDE.

Header File Generation

For all FUNCTION definitions which are not PRIVATE, a function map declaration will be generated. This will be an action within the IDE, facilitated by Proparse.

Function Forward within a Class File

Frequently, functions and procedures are defined in a group at the bottom of a source file, and function forward declarations are required to be at the top. One might want to use a header file for the top of the class file, but I propose an alternative approach. We would have a simple code generation action within the IDE which would generate a function forward declaration for a selected function. This way, the function forward declarations within the class file are painless both to generate as well as to maintain.

Run Time Errors

The problem with run-time resolution using *IN proc-handle* is that it frequently results in run-time errors. We will address this issue, both for functions and for procedures, by adding a Proparse facilitated extended xref database within the IDE. Automated code checks will be done by the IDE to ensure that the run-time errors will not happen.

Alternative Techniques

One technique uses a function as a wrapper for an internal procedure:

```
function {1} returns logical:
  def var retVal as logical no-undo.
  run getValue(output retVal) in {2}.
  return retVal.
end function.
```

We are interested in the ease of building automated tools for refactorings such as "Rename Method" and "Change Method Signature". In order for a refactoring tool to find and modify all external calls to a class's public method, the tool would have to figure out which function definitions are only wrappers for procedures – which is rather impractical, and, unnecessary.

Another reason for avoiding this technique is for the amount of code that this generates. For every inclusion of the header file, the code for a full function *definition* is generated. It is prefereable to only generate a function *declaration*, which is much smaller.

Another technique is to use a header file which uses function declarations but without using MAP TO:

function getName returns character in {1}.

The problem with this approach is with name collisions. There may be more than one class which would would like to "import", each with their own *getName* function. Additionally, we may wish to create more than one "object", in which case the single function name is unsufficient. This is why it is important to use MAP.

Calling Procedures

The problem within the IDE for providing code completions for procedures for a given handle is that within a RUN statement, the handle name comes after the method name – somewhat backwards from an OO perspective. I propose the following workaround. The programmer would type:

run myHandle

and at that point the code completions would be available for public procedures within *myObject*. Upon selection of a method name, the IDE would change the code to:

run theProcedure in myHandle

and further "context assist" would be available at that point for the procedures parameters.

John Green
john@joanju.com
+1-604-767-8587