Macro Pack for 4D 2004

Welcome to 4D 2004 Macro Pack!

Macro Pack is a component that contains seventeen macros designed to help increase your productivity when programming in the 4D 2004 environment! This guide describes each of the Macros in Macro Pack and demonstrates how to use them.

What is a Macro?

A macro is a section of 4D code that is permanently accessible and that can be inserted anywhere in your methods. Macros are stored in an XML format (text) file. They can contain all types of 4D text, commands and constants, as well as special tags which are replaced at the time of macro insertion by values derived from the method context. You can now use the 4th Dimension 2004 Method editor to generate and use macros that execute 4D project methods. This allows developers to create sophisticated functions broadcast using macro-commands linked to components. To call 4D code within a method, a new double tag has been added to the 4th Dimension 2004 macro language: **<method> </method>**. For more information on macros, please see the *4D 2004 Upgrade Reference Manual.*

Macro Pack Macros

Macro Name	Macro Function
Doc4DHeader	- Generates a comment block
Doc4D	- Generates API documentation for comment block
Modularize	- Helps to break down methods
TestMethod	- Tests the method
NameParameters	- Creates named variables and assigns parameters
AssignNamedParameters	- Assigns parameters to named variables
OrganizeDeclarations	- Organizes compiler declarations
MakeNewProcess	- Makes new process calls from project method calls
DeclareSOAPInput	- Generates soap declarations from compiler declarations
DeclareSOAPOutput	- Same as above but for output
XPathToSAX	- Generates SAX commands that creates XML documents
CVSInit	- Initializes cvs
CVSAdd	- Adds a method to the cvs repository
CVSOptions	- Sets cvs preferences
CVSCheckout	- Checks out source from the cvs repository
CVSRemove	- Removes a method from the cvs repository
CreateObject	- Examines COM objects, their members, and generates scripts that can access those objects from 4D.

Installing Macro Pack

The Macro Pack installer copies the Macro Pack macros into a folder named “Macros” which resides next to the “Macros.xml” file which is located in the active 4D folder of the machine.

The Macro Pack package comes with two distributions so there are two different ways to get started using Macro Pack.

1. Macro Pack Component

1. To install the component, you will need 4D Insider.

2. Create a new database using 4th Dimension or use an existing one and open it with 4D Insider.
3. From the menu bar, Select Components > Install/Update
4. Choose Macro Pack.4CP from the Macro Pack Component folder of the Macro Pack package.
5. Copy the folders named doc, Scripts, and Macros next to your data file.
6. Open your database with 4th Dimension and from User Mode, execute the method named MEM_InstallMacros.
7. Repeat the steps above on different databases with which you would like to use Macro Pack.

2. Macro Pack Source

The Macro Pack source can be used by developers to explore, create, or extend Macro Pack macros.

1. Open Macro Pack.4DB using 4th Dimension.
2. From User Mode, execute the method named MEM_InstallMacros.
3. The Macro Pack source database automatically generates the required files and folders so no further installation is required.

Once installation is complete, Macro Pack macros are ready to be used.

Accessing Macros in 4D 2004

4D 2004 lets you access Macros several ways:

Macros Button: The new 4D 2004 Macros button is located in the Method editor toolbar. Click the Macros button to open a pulldown list from which you can choose the macro you want to run.

Insert Macros Command: From the Method Editor, choose Method > Insert Macro. Or Right-click (Windows) or Ctrl-click (Macintosh) in the editing area of the Method Editor pane and choose Insert Macros.

Type Ahead: 4th Dimension automatically displays the Macro based on the first few characters you type in the editing area of the Method Editor. Type name of the macro or the first few letters of the macro name to display a list. When the correct macro is displayed, press the Tab key again to insert the Macro.

Working with Macro Pack Macros

Below you will find a description of each Macro in Macro Pack along with instructions on how to run them.

1. Doc4DHeader
Doc4Header allows you to quickly document a method in your database. It generates a chunk of commented text at the point where the cursor is placed in the Method editor. The comments contain useful information such as Author, Version, and a parameter list that is generated based on the compiler declarations at the time of the call to this macro.

For example, suppose the following Compiler declarations are contained in the method.

```
C_TEXT($1;$myText;$2)
```

Calling this macro generates the following comments:

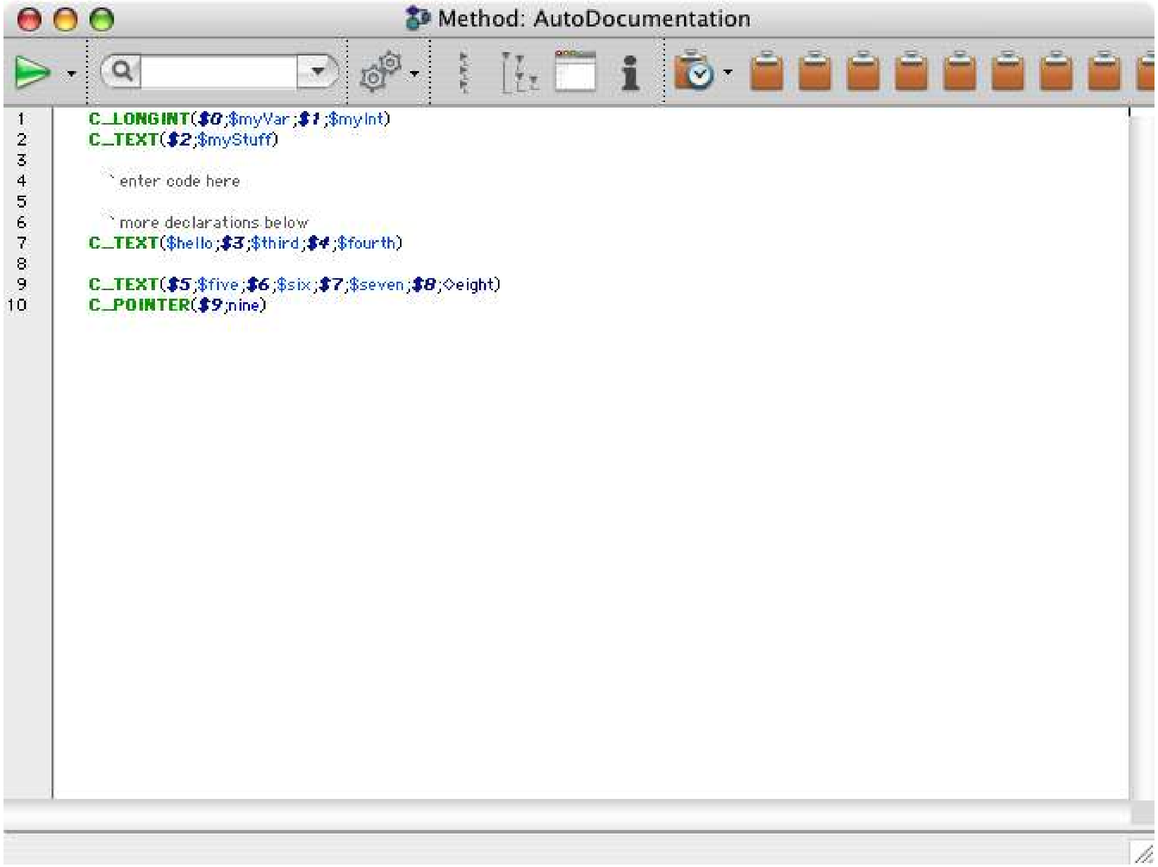
```
``
`
`
` Author: Jonathan Le
` Timestamp: 09/08/04 15:38:57
` Version:
`
` Parameters:
` $1 TEXT myText -
` $2 TEXT  -
``
```

Notice that the comment block starts and ends with double-grave characters (``). The documentation tags allow the author of the method to document important information like Author, Timestamp, and Version. The author can then fill this information by typing after the colon of each tag.

After the call to this macro, the cursor is placed immediately below the opening double-grave (``) characters.  The area below the (``) and before the first tag is considered the Description Area. This is where the author of a method describes what the method does. All formatting is kept intact.

To run Doc4Header:

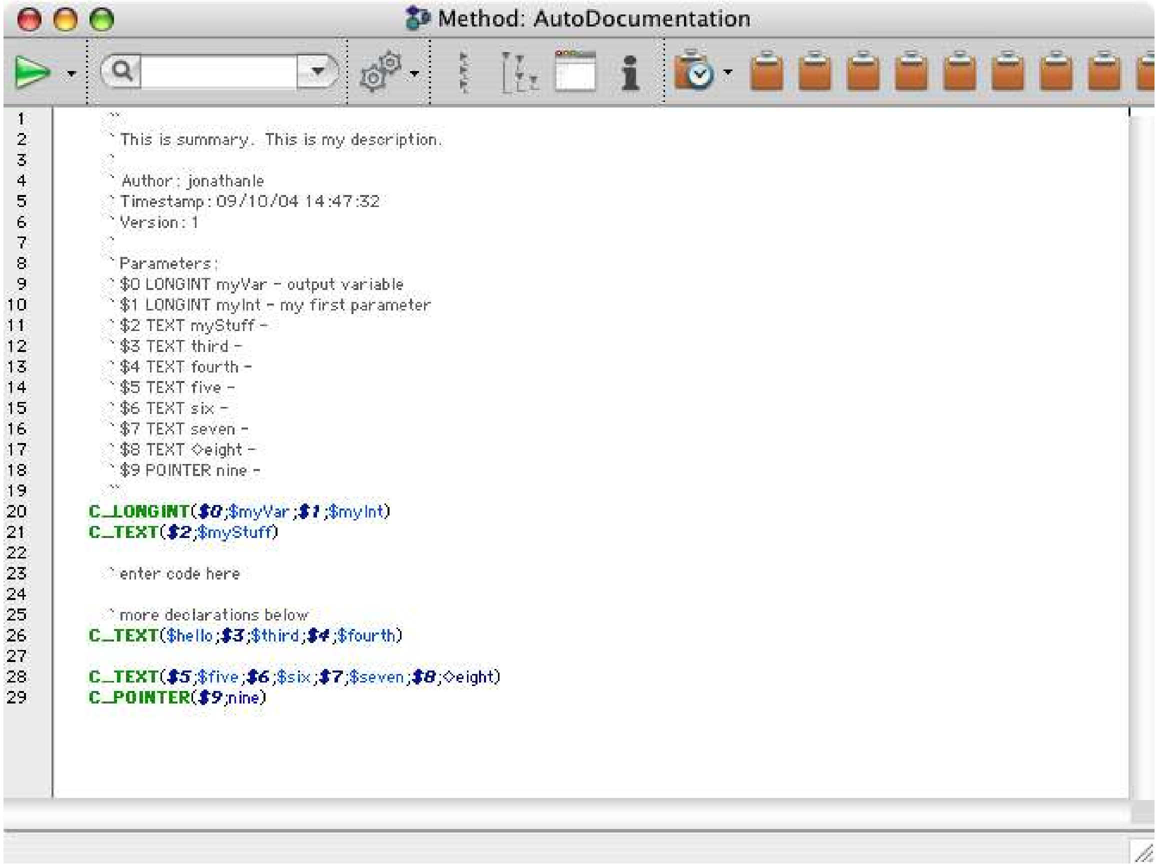
1. Simply set your cursor anywhere you want to generate the Doc4D comment block. You will most likely want to do this at the very beginning of your method.



```
C_LONGINT($0;$myVar;$1;$myInt)
C_TEXT($2;$myStuff)

  ` enter code here

  ` more declarations below
C_TEXT($hello;$3;$third;$4;$fourth)

C_TEXT($5;$five;$6;$six;$7;$seven;$8;◇eight)
C_POINTER($9;nine)
```

2. Run the macro called Doc4DHeader

3. The cursor is placed in the description area so you may type a short description of the current method.



```
   ``
   ` This is summary.  This is my description.
   `
   ` Author : jonathanle
   ` Timestamp : 09/10/04 14:47:32
   ` Version : 1
   `
   ` Parameters :
   ` $0 LONGINT myVar - output variable
   ` $1 LONGINT myInt - my first parameter
   ` $2 TEXT myStuff -
   ` $3 TEXT third -
   ` $4 TEXT fourth -
   ` $5 TEXT five -
   ` $6 TEXT six -
   ` $7 TEXT seven -
   ` $8 TEXT ◇eight -
   ` $9 POINTER nine -
   ``
C_LONGINT($0;$myVar;$1;$myInt)
C_TEXT($2;$myStuff)

   ` enter code here

   ` more declarations below
C_TEXT($hello;$3;$third;$4;$fourth)

C_TEXT($5;$five;$6;$six;$7;$seven;$8;◇eight)
C_POINTER($9;nine)
```

2. Doc4D

A call to the Doc4D macro generates API documentation using Doc4D comment blocks as described in Doc4Dheader above. You can use the Doc4DHeader to generate the comment blocks and enter the information you choose.

The following tags are allowed in Doc4D comment blocks:

Author:
Version:
Timestamp:
See:
Modified:

Each of these tags must appear on its own line after an opening double-grave (``). See Doc4DHeader above for an example. Note that each tag is a special word that Doc4D recognizes, so it must be separated by a space. For example,

```
` See: myMethodName
```

and not

```
` See:myMethodName
```

The parameter list is a list of all parameters available to the method. This list appears after the "Parameter" tag following the syntax:

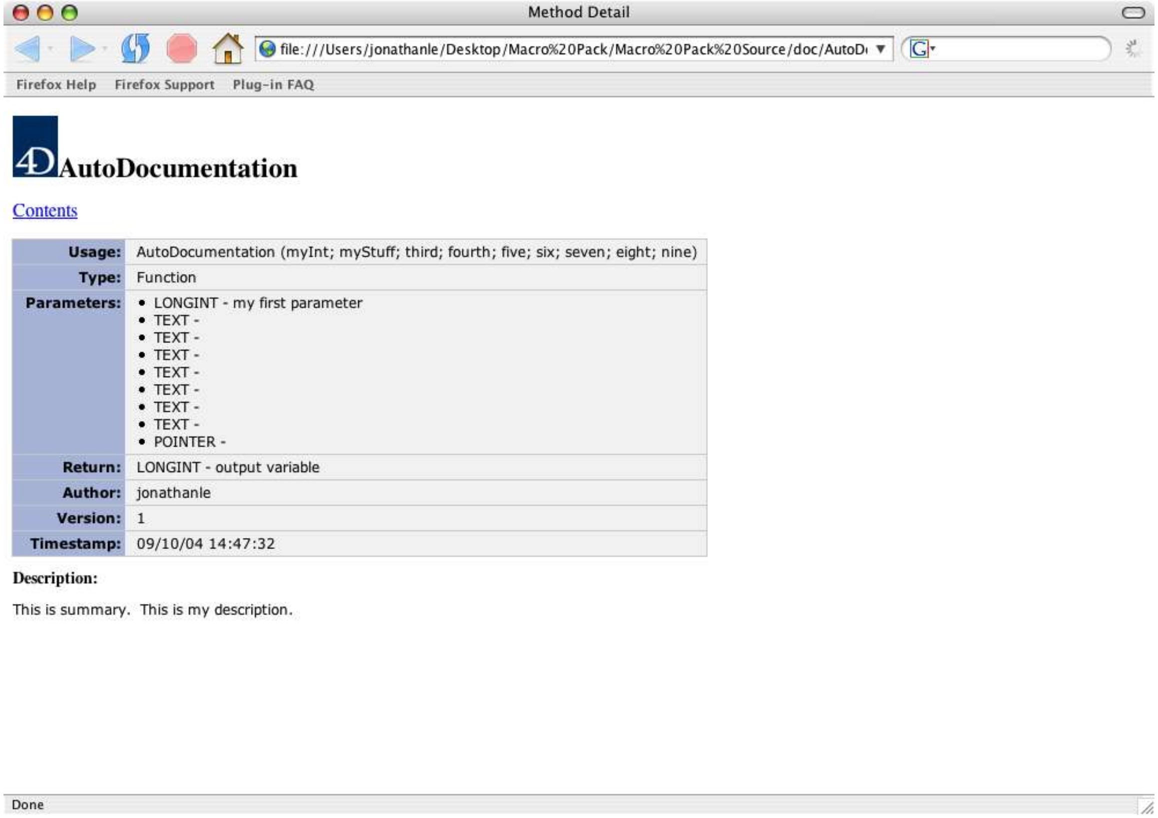
```
` $n TYPE variableName -
```

where n is the parameter number, TYPE is the parameter's type { TEXT | LONGINT | BOOLEAN | ... } and variableName is the named variable to which this parameter is assigned.

After the hyphen symbol (-) the user can enter the description for that parameter.

To run Doc4D Macro:

After a comment block has been either generated using Doc4DHeader or typed in manually, Doc4D can be run. It will generate an API documentation XML document in a special folder called "doc" that is placed next to your database's data file. This document can be opened using a browser with built in XML/XSL capabilities like Internet Explorer 6.0, Mozilla, Firefox. Safari currently does not yet provide XML and XSL support.


Method Detail
file:///Users/jonathanle/Desktop/Macro%20Pack/Macro%20Pack%20Source/doc/AutoD
Firefox Help
Firefox Support
Plug-in FAQ
Done

AutoDocumentation

Contents

Usage:	AutoDocumentation (myInt; myStuff; third; fourth; five; six; seven; eight; nine)
Type:	Function
Parameters:	• LONGINT - my first parameter • TEXT - • TEXT - • TEXT - • TEXT - • TEXT - • TEXT - • TEXT - • POINTER -
Return:	LONGINT - output variable
Author:	jonathanle
Version:	1
Timestamp:	09/10/04 14:47:32

Description:

This is summary. This is my description.

3. Modularize

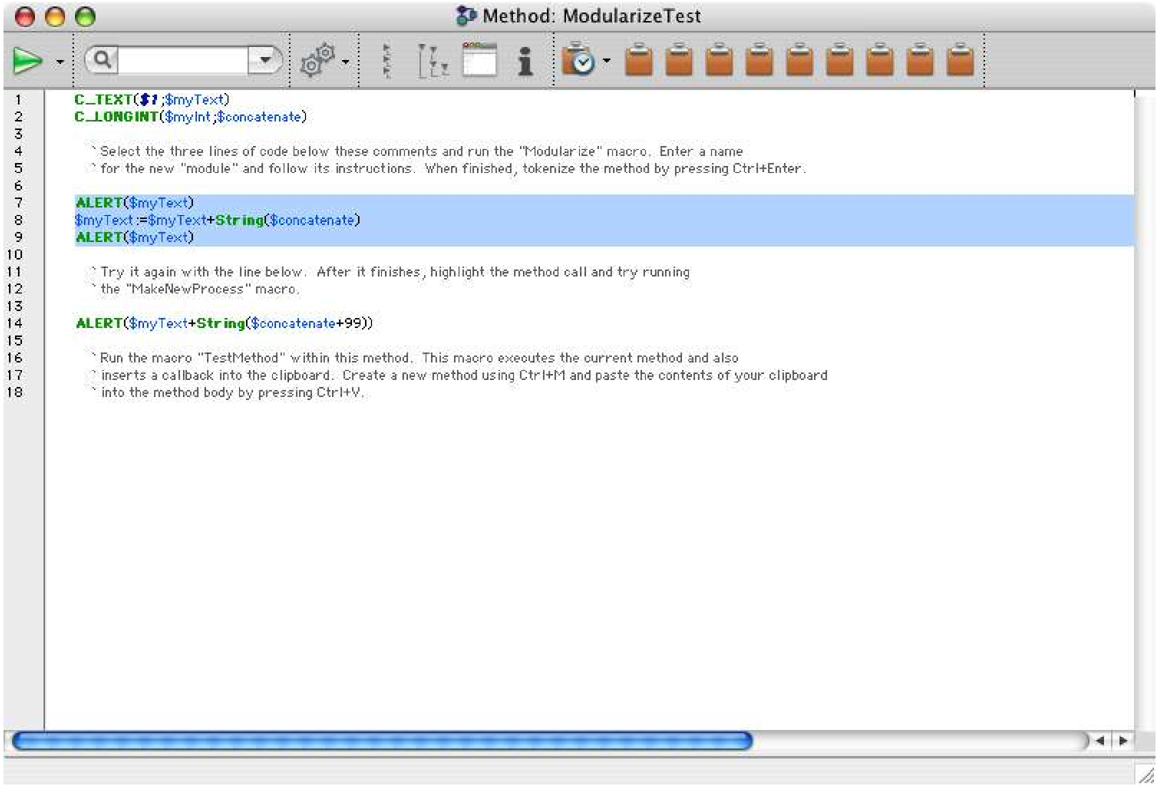
The Modularize macro helps break down large methods into smaller components. All the user has to do is highlight some chunk of code that needs to be modularized and run the macro.

The macro opens a dialog that allows the user to name a new method. Then, it will output a call to that method. All the user has to do is, from Design mode, create a method with the same name and paste the contents of the clipboard into that new method.

The macro catches all variables that are used in the chunk of code and attempts to re-declare them in the new method as parameters.

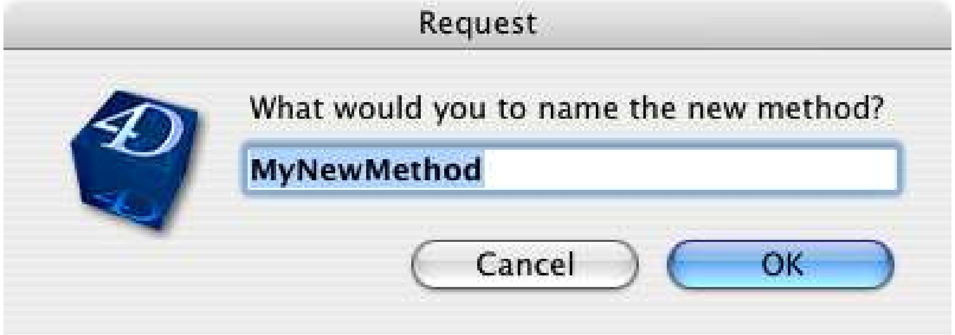
To run the Modularize macro:

1. Select a block of code from a method that you want broken down.



```
C_TEXT($1;$myText)
C_LONGINT($myInt;$concatenate)

  ` Select the three lines of code below these comments and run the "Modularize" macro.  Enter a name
  ` for the new "module" and follow its instructions.  When finished, tokenize the method by pressing Ctrl+Enter.

ALERT($myText)
$myText:=$myText+String($concatenate)
ALERT($myText)

  ` Try it again with the line below.  After it finishes, highlight the method call and try running
  ` the "MakeNewProcess" macro.

ALERT($myText+String($concatenate+99))

  ` Run the macro "TestMethod" within this method.  This macro executes the current method and also
  ` inserts a callback into the clipboard.  Create a new method using Ctrl+M and paste the contents of your clipboard
  ` into the method body by pressing Ctrl+V.
```

2. Run the Modularize macro. A message box will appear asking you to enter the name of the method.

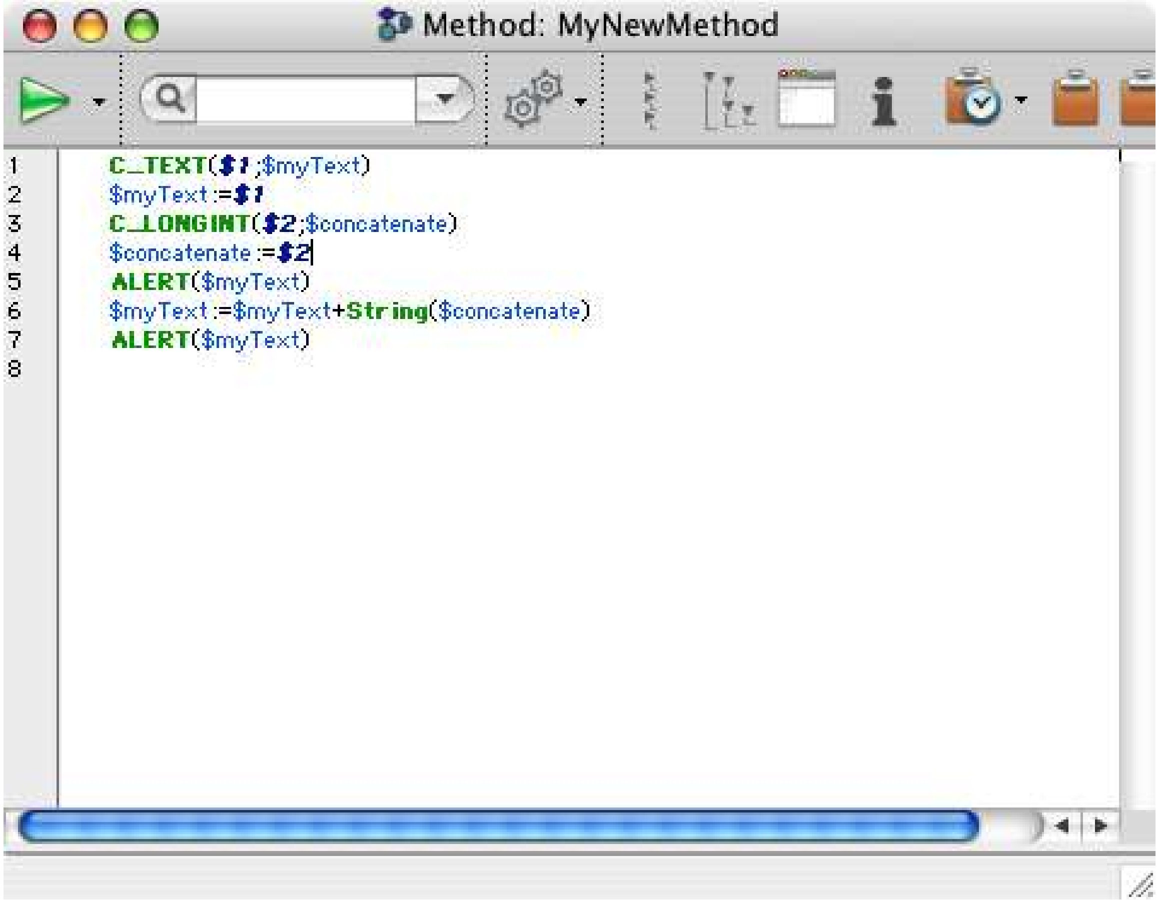


What would you to name the new method?

MyNewMethod

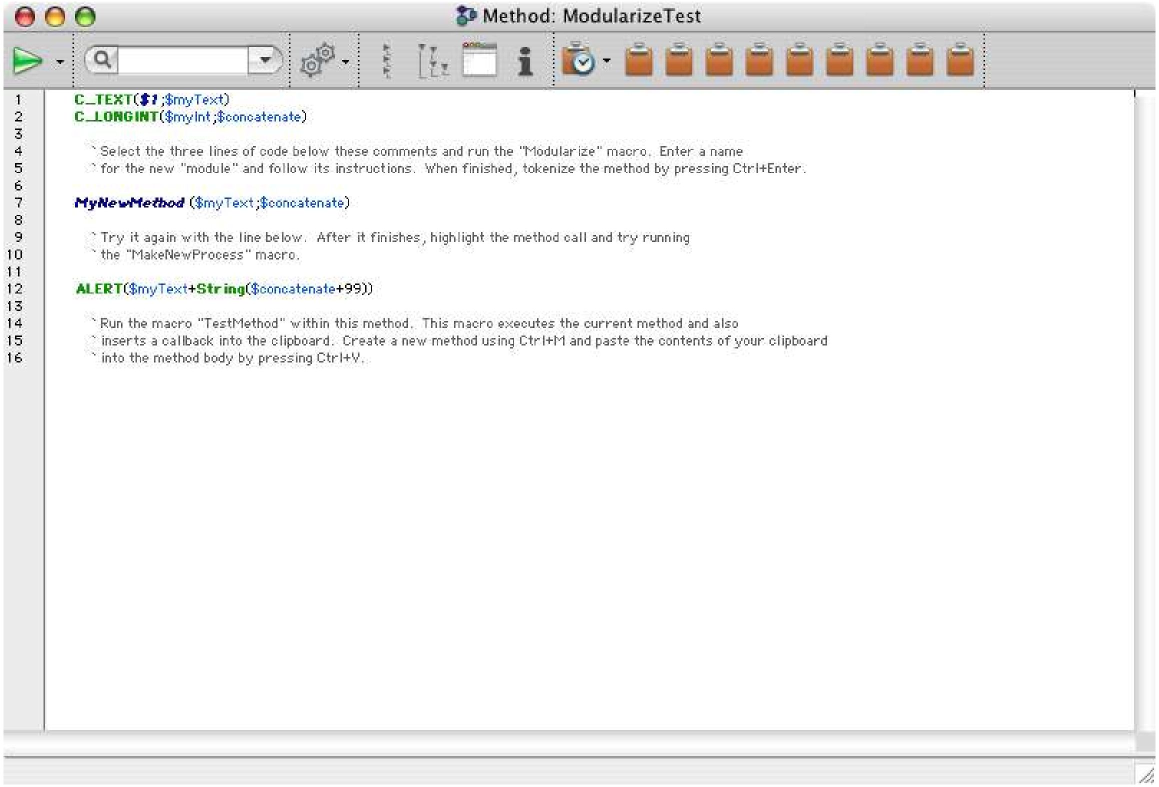
3. Create a method in 4D that matches the name of the method you entered in the dialog in step 2.

4. In the body of that new method, paste the contents of the clipboard.



```
C_TEXT($1;$myText)
$myText:=$1
C_LONGINT($2;$concatenate)
$concatenate:=$2
ALERT($myText)
$myText:=$myText+String($concatenate)
ALERT($myText)
```

You should have created a new method that contains your original code and a callback from the original method to the newly created one.

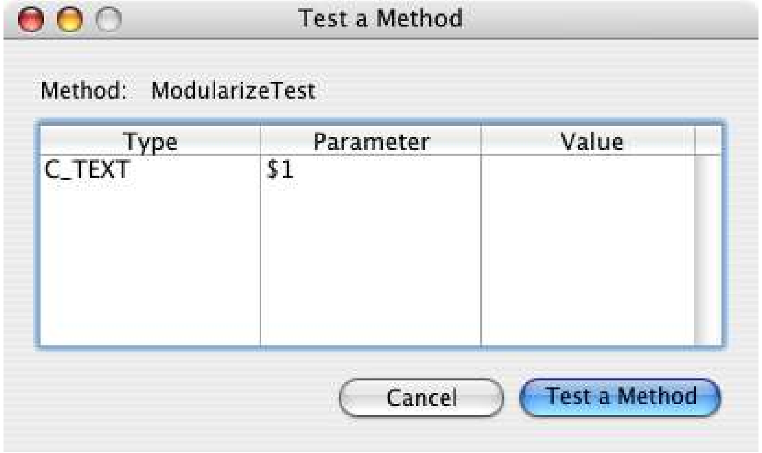


```
C_TEXT($1;$myText)
C_LONGINT($myInt;$concatenate)

  ` Select the three lines of code below these comments and run the "Modularize" macro.  Enter a name
  ` for the new "module" and follow its instructions.  When finished, tokenize the method by pressing Ctrl+Enter.

MyNewMethod ($myText;$concatenate)

  ` Try it again with the line below.  After it finishes, highlight the method call and try running
  ` the "MakeNewProcess" macro.

ALERT($myText+String($concatenate+99))

  ` Run the macro "TestMethod" within this method.  This macro executes the current method and also
  ` inserts a callback into the clipboard.  Create a new method using Ctrl+M and paste the contents of your clipboard
  ` into the method body by pressing Ctrl+V.
```

4. TestMethod

This macro simply prompts the user with a dialog asking to input some values for each declared parameter, then runs the method. Next, it copies a call back to the method with the same values passed to the clipboard. A user is then able to paste that into a new method that can be used as a method for running test suites.

To run this macro, simply open the method you want to test and run the TestMethod macro. A dialog appears asking you to enter values for any parameters you may have declared in that method.



Method: ModularizeTest

Type	Parameter	Value
C_TEXT	$1	

Pressing the "Test a Method" button will execute the method using the values you supply as arguments. In addition, your clipboard is filled with a callback to that method with the arguments you entered.

5. NameParameters & AssignNamedParameters

AssignNamedParameters takes some compiler declarations of the form

```
C_TYPE($1;$varName1;...;$n;$varNameN)
```

where $n is the number of parameters and $varname is any variable name, and assigns the parameters to the variables following. For example,

```
C_TEXT($1;$myText;$0;$output)
```

will generate

```
$myText:=$1
$0:=$output
```

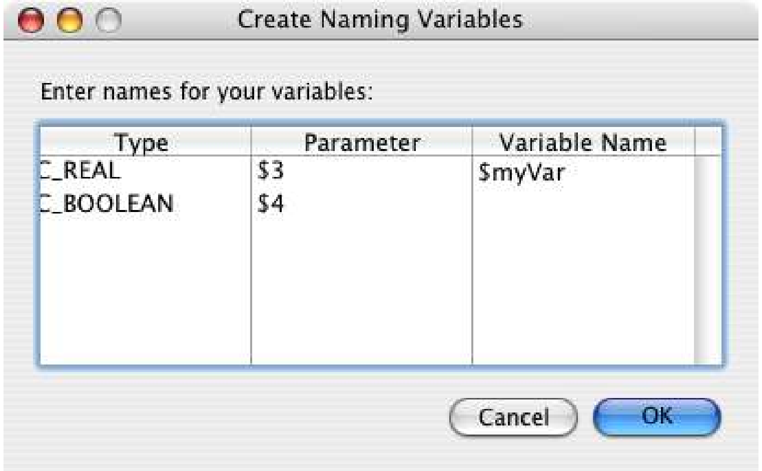
Likewise, NameParameters takes compiler declarations without a following named variable and generates assignments by asking the user to enter a variable name.

```
C_TEXT($1)
```

would then generate

```
$myText:=$1
```

provided the user entered $myText in the dialog prompt.



Type	Parameter	Variable Name
C_REAL	$3	$myVar
C_BOOLEAN	$4	

6. OrganizeDeclarations

This macro will organize cluttered compiler declarations and group them by type, declare them at the top of the selection, and leave the rest of the code intact. For example:

```
C_TEXT($a;$b)

ALERT($a)

C_TEXT($c)
```

would be reorganized to

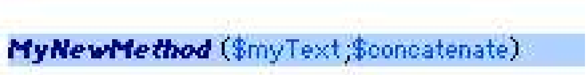
```
C_TEXT($a;$b;$c)

ALERT($a)
```

Note: the declarations will be grouped as long as the variables are declared in sequential order.

7. MakeNewProcess

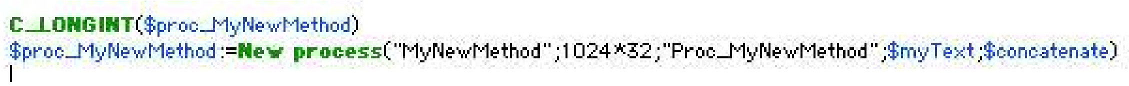
This macro replaces a project method call with a call to New Process passing in the call to the project method.

To use this method, you only have to highlight a call to a project method.



```
MyNewMethod ($myText;$concatenate)
```

Run the macro named MakeNewProcess. This should change the call to that same project method into a call to the project method but in a new process, preserving any arguments that you may have passed.



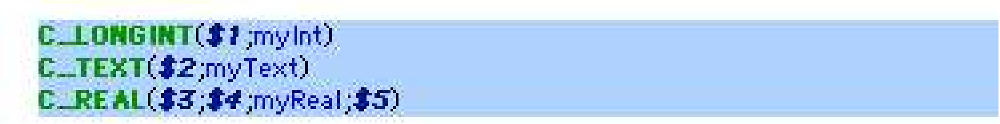
```
C_LONGINT($proc_MyNewMethod)
$proc_MyNewMethod:=New process("MyNewMethod";1024*32;"Proc_MyNewMethod";$myText;$concatenate)
```

8. DeclareSOAPInput/DeclareSOAPOutput

Given some compiler declarations, this macro generates SOAP declarations of either input or output. This is especially useful for transforming regular methods into web service methods.

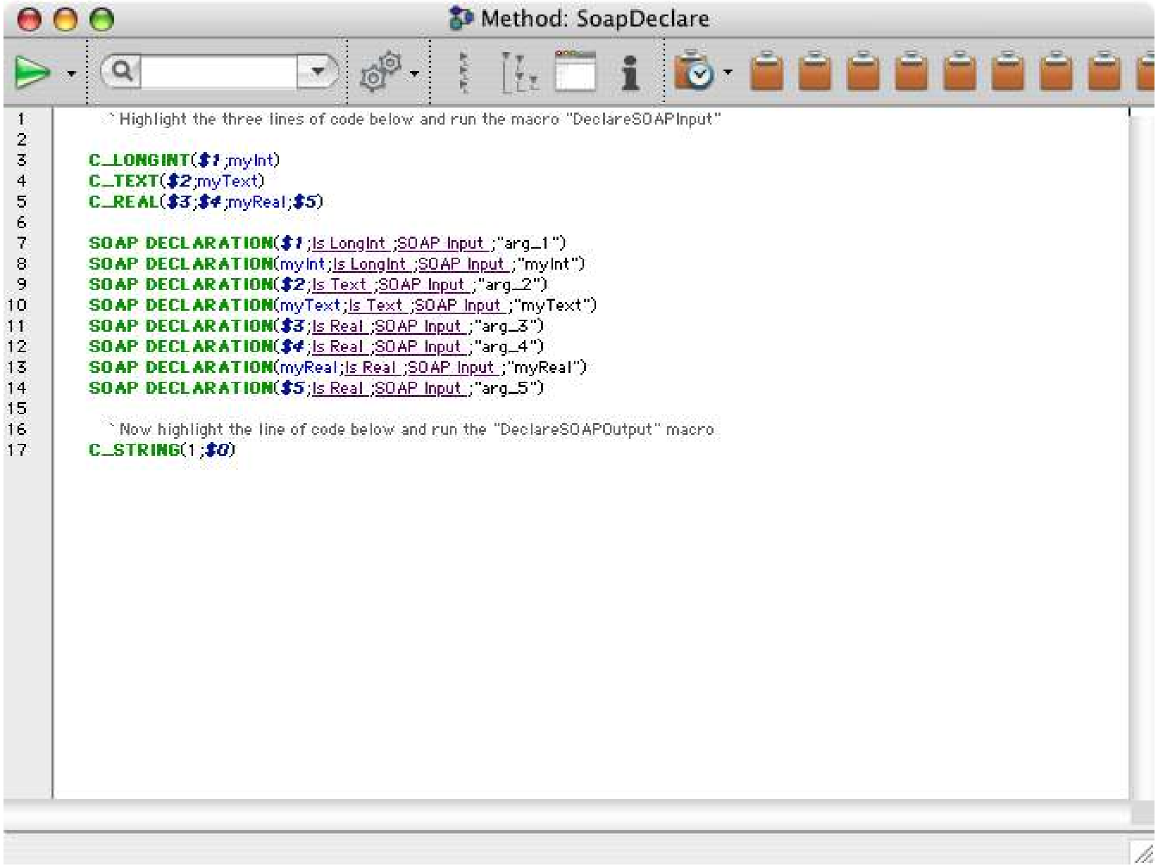
To run DeclareSOAP:

1. To start, select one or more compiler declarations from a method.



```
C_LONGINT($1;myInt)
C_TEXT($2;myText)
C_REAL($3;$4;myReal;$5)
```

2. Next run the DeclareSOAPInput or DeclareSOAPOutput macro to create input or output soap declarations.



```
  ` Highlight the three lines of code below and run the macro "DeclareSOAPInput"

C_LONGINT($1;myInt)
C_TEXT($2;myText)
C_REAL($3;$4;myReal;$5)

SOAP DECLARATION($1;Is LongInt ;SOAP Input ;"arg_1")
SOAP DECLARATION(myInt;Is LongInt ;SOAP Input ;"myInt")
SOAP DECLARATION($2;Is Text ;SOAP Input ;"arg_2")
SOAP DECLARATION(myText;Is Text ;SOAP Input ;"myText")
SOAP DECLARATION($3;Is Real ;SOAP Input ;"arg_3")
SOAP DECLARATION($4;Is Real ;SOAP Input ;"arg_4")
SOAP DECLARATION(myReal;Is Real ;SOAP Input ;"myReal")
SOAP DECLARATION($5;Is Real ;SOAP Input ;"arg_5")

  ` Now highlight the line of code below and run the "DeclareSOAPOutput" macro
C_STRING(1;$0)
```

Note that the soap declarations appear immediately below the compiler declarations once the macro finishes. The parameters are labeled with "arg_N," where N is the number of the parameter.

9. XPathToSAX

Using this macro will generate code to create an XML document based on a given XPath.

Take the following XML structure as an example:

```
<a>
 <b>
  <c/>
 </b>
</a>
```

To create this XML document, you can use XPathToSAX to generate 4D code that would produce this document. All you have to provide is the XPath to the deepest element in an XML structure.

In the example, the deepest element is "c." And the XPath notation to this element is

```
/a/b/c
```

From the method editor, type in the XPath as shown above. Highlight the XPath, then run the macro XPathToSAX. What is generated is 4D code that looks something like the following:

```
C_TIME($document)
$document:=Create document("XML_Filename";"XML")

SAX OPEN XML ELEMENT($document;"a")

SAX OPEN XML ELEMENT($document;"b")

SAX OPEN XML ELEMENT($document;"c")
SAX CLOSE XML ELEMENT($document)

SAX CLOSE XML ELEMENT($document)

SAX CLOSE XML ELEMENT($document)

CLOSE DOCUMENT($document)
```

Now all you have to do is rename the XML file.

10.CVS

CVS stands for Concurrent Versions System. CVS has a number of different uses including source code management, backup, and of course versioning. The system is particularly useful when there are a large amount of source files in a project and when there are more than one developer working on the project. It also allows you to “branch out,” and create different releases of a product (alpha, beta, production, etc) and makes it easy to “roll back” in case a certain release does not pass testing.

When we use CVS with 4D, we are only interested in the source code of project methods. The project methods can be exported into C4D format which can be added and committed into a version control system like CVS. This would involve going to the menu bar, exporting to a C4D file, then from CVS adding it through by typing CVS commands in the command line. We simplify this process by providing you with CVS macros:

- CVSOptions - This macro simply prompts the user with a dialog that allows them to set preferences like cvs executable* and repository location.

- CVSInit - initializes a repository for storing source files, then requests the user to perform an initial check out in a working directory.

- CVSAdd - adds the current method to the repository by outputting a C4D method file and committing it to the repository.

- CVSRemove - removes the currently selected method from the repository and deleting its corresponding C4D file from the working directory.

- CVSCheckout - Checks out a copy of the source files from the repository into a new working directory.

*Note: the CVS Executable needs to be the command-line version of the CVS tool. For both Mac and Windows, this executable can be retrieved from http://www.cvshome.org.

To set up CVS for Mac:

1. First, make sure Xcode 1.0 or above is installed.
2. Download and set up fink from http://fink.sourceforge.net
3. After fink has been completely installed and configured, run the following commands from the command line:

```
% sudo apt-get update
% sudo apt-get install cvs
```

Once successfully installed, CVS is ready for use and does not need to be further configured from Macro Pack.

To set up CVS under Windows:

1. Download WinCVS from http://www.wincvs.org
2. Use the CVSOptions macro to open the CVS preferences dialog. Make sure you point to the cvs executable contained in CVSNT folder of the WinCVS installation directory.

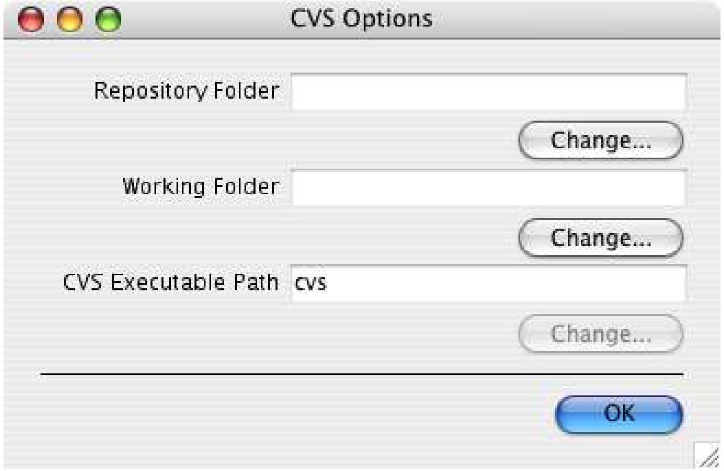
Once CVS has been installed and configured on your machine:

The first thing you would need to do is run the macro named CVSOptions. This prompts you with a dialog that you can use to select specific required paths such as:

- The working directory: This is where all your source files will be contained following a CVSCheckout, CVSAdd and from which files will be removed using CVSRemove.

- The repository: This is the location of your CVS repository. This folder is where all source files get committed in and out. Essentially, this is where source code is backed up and tracked using CVS.

- The CVS executable path: On Macintosh, this can be ignored. However, under Windows, you will need to point to the executable file of the CVS command-line tool. This is usually located under c:\WinCVS\CVSNT\cvs.exe assuming your installation directory was c:\WinCVS.

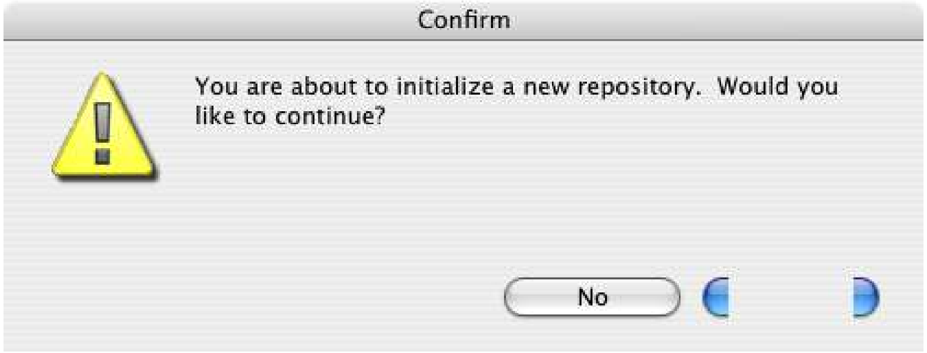


CVS Executable Path: cvs

Next you will need to run the macro called CVSInit. The macro will initialize your repository as well as perform the initial checkout. Once your repository has been initialized, you can freely Add or Remove methods.

To check out additional copies of your source, use the CVSCheckout macro. You can either use the working directory you initially used during setup, or you can check out the source to a new one.

Adding and removing using CVSAdd and CVSRemove immediately commits your method into or out of the repository. The method's body is saved in C4D format.



You are about to initialize a new repository. Would you like to continue?

11. CreateObject (available to Windows users only)

COM objects are code libraries that expose methods for world access. For example, Microsoft Outlook has includes an Outlook COM object that exposes methods that allow a user to add a contact, add a meeting, query for an appointment, and so on.

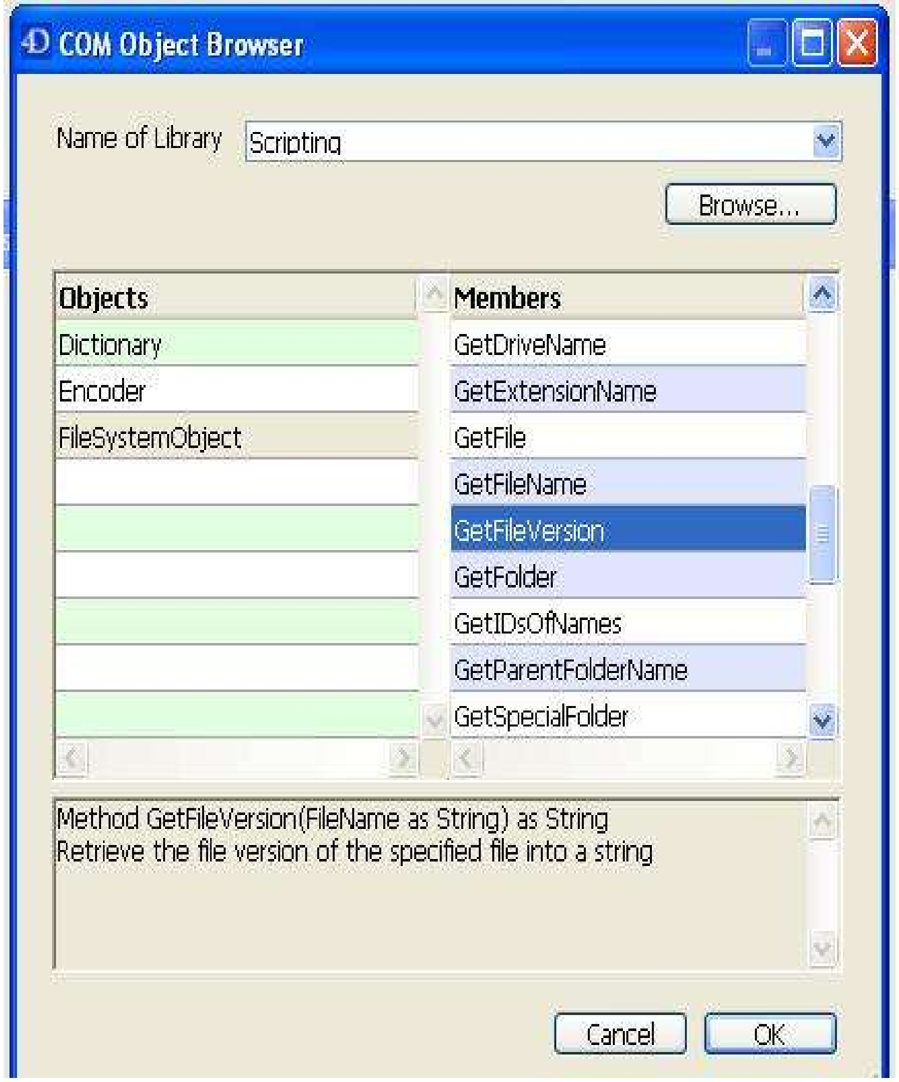
That said, the CreateObject macro allows you to browse through objects and their members (methods and properties) and create 4D project methods that can access them. It does this by dynamically generating VBScripts based on your selections and using a command called LAUNCH EXTERNAL PROCESS to execute them.

To use CreateObject:

1. Create a new method. In this example we'll name it "COMObjects."

2. Run the CreateObject macro. A dialog listing some Microsoft Windows pre-bundled COM objects should be listed. Clicking on one of the objects will display all its members and clicking on a member will display detailed information about that member.

3. Select an Object as well as a member and note its parameters:

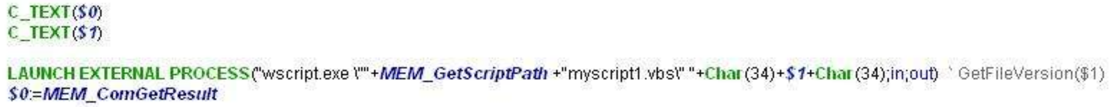


Name of Library: Scripting

Objects	Members
Dictionary	GetDriveName
Encoder	GetExtensionName
FileSystemObject	GetFile
	GetFileName
	GetFileVersion
	GetFolder
	GetIDsOfNames
	GetParentFolderName
	GetSpecialFolder

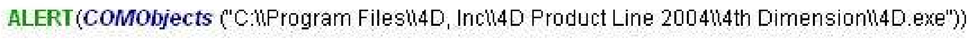
Method GetFileVersion(FileName as String) as String
Retrieve the file version of the specified file into a string

4. When you are satisfied, click the "OK" button. The macro will generate a script based on your selection and save it in the Scripts folder located next to your database's data file. It also generates code that executes the script.



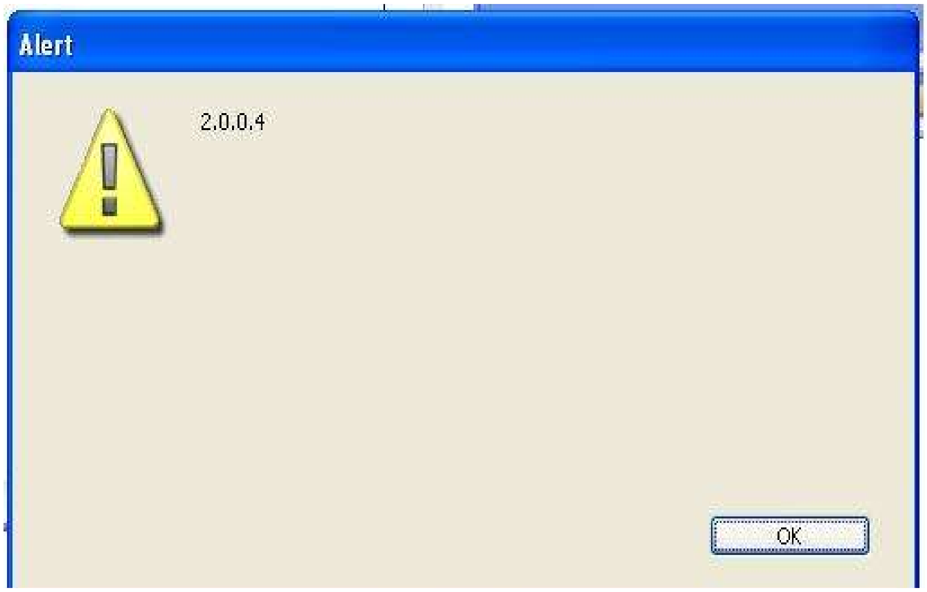
```
C_TEXT($0)
C_TEXT($1)

LAUNCH EXTERNAL PROCESS("wscript.exe \""+MEM_GetScriptPath +"myscript1.vbs\" "+Char(34)+$1+Char(34);in;out)  ` GetFileVersion($1)
$0:=MEM_ComGetResult
```

4. Now, from another method, you can call the new COM object method and pass it the same parameters as required by the COM object:



```
ALERT(COMObjects ("C:\\Program Files\\4D, Inc\\4D Product Line 2004\\4th Dimension\\4D.exe"))
```

5. Test the method and verify your results:



2.0.0.4

Additional Resources:

1. Macro Pack Source

Each of these macros can be found within the following three XML files:

MethodEditorMacros.xml
ComObjectMacros.xml
CVSMacros.xml

To start modifying methods that are called by these macros, explore each of these xml files that can be identified by looking in between the <method></method> tags.

You can then use the Macro Pack Source to modify or add on to these methods. For example, the Doc4D macro uses the DOC_Doc4D method which can be found in the Method Editor Macros > AutoDocumenter Methods group from the Home theme of the Explorer.

2. CVS

For more information on CVS, please refer to http://www.cvshome.org

There is also a graphical user interface for CVS which can be found on the following sites:

Windows users: http://www.wincvs.org
Mac users: http://www.maccvs.org