# CodeWarrior™ Development Studio

## IDE 5.6 Windows® Automation Guide

Revised: May 30, 2006

freescale™ semiconductor

Freescale, the Freescale logo, and CodeWarrior are trademarks or registered trademarks of Freescale Corporation in the United States and/or other countries. All other trade names and trademarks are the property of their respective owners.

Copyright © 2006 by Freescale Semiconductor company. All rights reserved.

**No portion of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Freescale. Use of this document and related materials is governed by the license agreement that accompanied the product to which this manual pertains. This document may be printed for non-commercial personal use only in accordance with the aforementioned license agreement. If you do not have a copy of the license agreement, contact your Freescale representative or call 1-800-377-5416 (if outside the U.S., call +1-512-996-5300).**

Freescale reserves the right to make changes to any product described or referred to in this document without further notice. Freescale makes no warranty, representation or guarantee regarding the merchantability or fitness of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product described herein and specifically disclaims any and all liability. **Freescale software is not authorized for and has not been designed, tested, manufactured, or intended for use in developing applications where the failure, malfunction, or any inaccuracy of the application carries a risk of death, serious bodily injury, or damage to tangible property, including, but not limited to, use in factory control systems, medical devices or facilities, nuclear facilities, aircraft navigation or communication, emergency systems, or other applications with a similar degree of potential hazard.**

## How to Contact Freescale

| | |
|---|---|
| **Corporate Headquarters** | Freescale Corporation<br>7700 West Parmer Lane<br>Austin, TX 78729<br>U.S.A. |
| **World Wide Web** | http://www.freescale.com/codewarrior |
| **Technical Support** | http://www.freescale.com/support |

# Table of Contents

1

# Getting Started

This manual describes how to use external applications and scripting environments to automate the CodeWarrior™ IDE to perform certain tasks such as manipulating CodeWarrior projects, building targets, compiling and linking project files, debugging projects, displaying IDE messages, and using version control features in the IDE.

This chapter has these sections:

- Overview of This Manual on page 5
- Related Documentation on page 6

## Overview of This Manual

This manual contains information specific to CodeWarrior IDE automation. Table 1.1 on page 5 describes the information contained in each chapter in this manual.

**Table 1.1 Contents of chapter**

| Chapter | Description |
|---|---|
| Getting Started on page 5 | (this chapter) |
| IDE Command-Line Scripting on page 9 | describes use of the CodeWarrior IDE command-line executable program, and provides a reference to all of its' command-line switches |

**Table 1.1 Contents of chapter (*continued*)**

| Chapter | Description |
|---|---|
| Tcl and Command Window Scripting on page 13 | describes how to control the CodeWarrior IDE using the Tcl scripting language, describes the CodeWarrior Command Window, and provides a reference of Command Window options |
| Microsoft COM Automation on page 51 | describes how to use Component Object Model (COM) objects that the IDE exposes and the methods you can call to work with those objects using the OLE/COM Object Viewer |

## Related Documentation

This section describes the supplementary CodeWarrior documentation, third-party documentation, and references to helpful code examples and web sites.

### IDE Command-Line Tools

This manual only describes one of many components of the CodeWarrior command-line tool set. For information about other CodeWarrior command-line tools, refer to the *C Compilers Reference* manual and the *CodeWarrior IDE User's Guide*.

### Tcl Scripting

For in-depth information about the Tcl scripting language, refer to the *Tcl8.3/Tk8.3 Manual* in the CodeWarrior help system or the Tcl web site:

```
http://www.tcl.tk
```

**NOTE** Command hints and short command forms are not available for built-in Tcl commands.

### Perl Scripting

For in-depth information about the Perl scripting language, refer the Perl web site:

`http://www.perl.org`

You can find the latest version of Perl at:

`http://www.cpan.org/src`

### VBScript Scripting

For in-depth information about the VBScript scripting language, refer this URL:

`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vtoriVBScript.asp`

### Microsoft COM Automation

To control the IDE, your Perl/VBScript scripts must manipulate the IDE’s COM objects. You can get a copy of Microsoft’s OLE/COM Object Viewer at:

`http://www.microsoft.com/com/resources/oleview.asp#OLEViewer`

Also, to manipulate COM objects through Perl, you need the Win32::OLE module. You might also want to use the other WIN32::OLE modules (such as Win32::OLE::Enum). You can get these modules at:

`http://www.cpan.org`

2

# IDE Command-Line Scripting

This chapter describes how to control the CodeWarrior™ IDE with the command-line executable program.

This chapter has these sections:

## Overview

The CodeWarrior IDE provides command-line access to different components of the IDE. You access the components by executing command-line tools. This chapter focuses specifically on the CodeWarrior IDE command-line tool.

The CodeWarrior IDE command-line tool allows you to instruct the IDE to manipulate and build projects, compare source files, run Tcl scripts, and obtain the version of the IDE. The IDE command-line tool for Windows host is an executable program named `CmdIDE.exe`, and is located in the following directory: `{CodeWarrior_Dir}\bin`, where `{CodeWarrior_Dir}` is the directory where you installed the CodeWarrior development tools.

**NOTE** Command-line compiler, linker, and debugger tools may be available on your particular platform. Refer to the *C Compilers Reference* manual for information about other command-line tools available on your platform.

You interact with the command-line tools through a text-based console or terminal rather than a graphical user interface. You specify command-line tool options in the command line you use to invoke a tool. Command-line options you specify on the command-line are also called *switches*.

## Running the IDE Command-Line Tool

The IDE command-line tool performs operations on files you specify on the command line. If the tool successfully finishes its operation, a new prompt appears on the command line. Otherwise, it reports any problems as text messages on the command line before a new prompt appears.

You can also write scripts that automate the process to build your software. Scripts contain a list of commands and command-line tools to invoke, one after another.

For example, the `make` tool, a common software development tool, uses scripts to manage dependencies among source code files and invoke command-line compilers, assemblers, and linkers as needed, much like the CodeWarrior project manager.

## IDE Command-Line Tool Reference

This section lists the various operations and switches you can use to cause the IDE to perform certain tasks.

The syntax for invoking the IDE command-line tool on Windows is:

```
cmdIDE [[files...] [function [options...] ...]]
```

The *`files`* parameter is a list of zero or more files on which the IDE should operate. The IDE processes files in the order you specify them in the command line. If you specify one or more CodeWarrior project files, the first project file in the command line is the CodeWarrior default project.

The *`function`* parameter is the operation you want the IDE to perform. You may specify multiple functions in a single command line for the IDE to perform.

The *`options`* parameter is a list of zero or more command-line switches that tell the IDE how to perform the specified operation. If you use a switch that is inappropriate for an operation, the IDE ignores the switch and completes processing of all other switches.

The files, options, and switches you specify in the command line depends on the operation you want the IDE to perform. The rest of this section describes the various functions and corresponding switches the IDE understands.

## Startup Operations

The startup operations lets you instruct the IDE to start the CodeWarrior IDE and run the specified script. Table 2.1 on page 11 describes the parameters for the startup operations.

**Table 2.1 Startup Operations Command-Line Parameters**

| Switch | Description |
|---|---|
| `/f` | makes the IDE the front window<br>options:<br>• y - focus to IDE (default)<br>• n - start minimized |
| `/s` | forces the command line to be processed in a new instance of the IDE instead of using the current IDE instance |
| `/x <project.xml>` | specifies an XML project file to import |
| `/w <ws.cww>` | starting a new workspace<br>• n - start with no workspace |

## Build Operations

The build parameters lets you instruct the IDE to build projects. Table 2.2 on page 12 describes the build function parameters.

**Table 2.2 Build Operations Command-Line Parameters**

| Switch | Description |
|---|---|
| `/v` | convert the project on opening<br>options:<br>• y - convert without asking<br>• n - do not convert<br>• a - ask before converting |
| `/t` *`targetname`* | switch the default target to the target named *targetname* |
| `/b` | build the current target |
| `/r` | remove object code from the default build target |
| `/c` | close the default project after the build is complete |
| `/q` | quit the IDE after the build is complete |

## Debug Operation

The debug parameters lets you instruct the IDE to start the Command-line debugger and run the specified script.

### Syntax

```
cmdIDE /d scriptfile
```

### Parameters

`scriptfile`

Supply the name or fully-qualified path to a Command Window script file (see “Tcl and Command Window Scripting” on page 13).

## Help Function

The help function causes the IDE to print a summary of all command-line arguments to the terminal.

### Syntax

```
cmdIDE /?
```

3

# Tcl and Command Window Scripting

You can control IDE functions with Tcl commands and Tcl-based Freescale commands. You can run these commands either interactively through the IDE **Command Window**, or write them in a script file that you can run interactively or specify as a IDE command-line parameter.

This chapter has these sections:

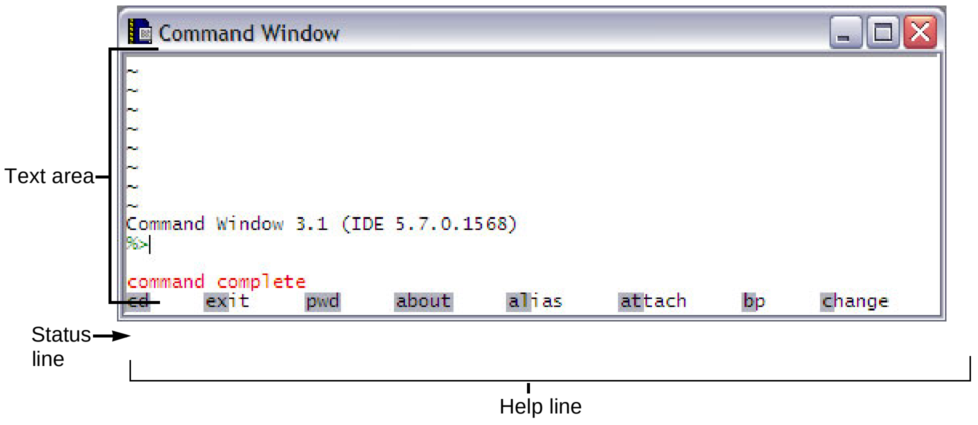
- Command Window Interface on page 13
- Running Tcl Scripts on page 16
- Tcl Built-in Commands on page 16
- Freescale Commands on page 17

## Command Window Interface

The **Command Window** (Figure 3.1 on page 14) is a window in the CodeWarrior™ IDE that lets you interactively execute Tcl commands.

To access this window in the Windows-hosted IDE, select **View > Command Window** from the menu bar. To access this window in the Linux- or Solaris-hosted IDE, select **Window > Command Window** from the menu bar.

### Figure 3.1 The Command Window



```
~
~
~
~
~
~
~
~
Command Window 3.1 (IDE 5.7.0.1568)
%>|

command complete
cd      exit    pwd     about     alias     attach     bp     change
```

Table 3.1 on page 14 describes each of the three parts of the **Command Window**.

### Table 3.1 Command Window Parts

| Part | Description |
|---|---|
| Text area | displays the command prompt, `%>`, and the text output of commands |
| Status line | displays the status of the last executed command |
| Help line | displays command hints for the Freescale commands |

To browse through all available command hints, press the space bar at an empty command prompt. The highlighted characters represent the short form of the command.

Command hints and short command forms are not available for built-in Tcl commands. Documentation for these commands is located in the *Tcl8.3/Tk8.3 Manual* in the CodeWarrior Help System, and at the Tcl web site:

`http://www.tcl.tk`

## Issuing Commands

To issue a Tcl command, type the command at the command prompt (`%>`). For Freescale commands, you may type either the normal or the short form of the command. If you specify a short-form command, pressing space or tab will auto-complete it.

## History Functions

To repeat the last command entered, press Enter on your keyboard. To browse the command history, press the up arrow or down arrow keys.

## Scroll Functions

To scroll the text area of the Command Window:

- Press the page up or page down keys on your keyboard to scroll the text area by the number of lines set with the `config` command. The default value is the number of lines currently displayed. This value is updated when you resize the Command Window.
- Press Control-up arrow or Control-down arrow on your keyboard to scroll the text window up or down by one line.
- Press Control-left arrow or Control-right arrow on your keyboard to scroll the text window left or right by one character.

## Copy and Paste Functions

To copy portions of the text window to the clipboard, hold down the left mouse button and drag the selection box around the desired text. Press Enter on your keyboard or select **Edit** > **Copy** from the CodeWarrior menu bar.

To paste text from the clipboard into the text area of the Command Window, click the left mouse button or select **Edit** > **Paste** from the CodeWarrior menu bar.

## Running Tcl Scripts

The built-in Tcl command `source` lets you run a sequence of Tcl commands that you have placed into a text file.

The command-line IDE lets you specify a Tcl script as a parameter. This makes it possible to run Tcl scripts from the system command-line without first opening the IDE Command Window.

Each time you open the **Command Window**, the IDE searches for a script file named `tcld.tcl` in the (`%SystemRoot%`) directory. If the IDE finds this script file, the IDE attempts to run it. Place commands into this script file that you want the IDE to run each time you open the Command Window or run a Tcl script.

**TIP** By convention, Tcl script files have the filename extension `.tcl`.

## Tcl Built-in Commands

The Tcl built-in commands are documented in the *Tcl8.3/Tk8.3 Manual*, located within the CodeWarrior Help System.

To display the version of your Tcl interpreter, type this command into the **Command Window**:

```
puts [info tclversion]
```

You can obtain additional information about Tcl here:

`http://www.tcl.tk`

## Freescale Commands

There are numerous Freescale commands that you may use within Tcl scripts or in the **Command Window**. This section describes each of these commands.

**NOTE** Shortcut command syntax (if available) is listed first, followed by formal syntax.

The commands are:

**Table 3.2 Freescale Commands**

| | | |
|---|---|---|
| alias on page 17 | bp on page 18 | bringtofront on page 20 |
| cd on page 21 | change on page 22 | cls on page 23 |
| config on page 24 | copy on page 27 | debug on page 27 |
| dir on page 28 | disassemble on page 29 | display on page 29 |
| evaluate on page 31 | exit on page 32 | getpid on page 33 |
| go on page 33 | help on page 34 | history on page 34 |
| kill on page 35 | log on page 35 | make on page 36 |
| next on page 36 | project on page 37 | pwd on page 37 |
| quitIDE on page 37 | radix on page 38 | removeobj on page 39 |
| reset on page 39 | restart on page 40 | restore on page 41 |
| save on page 41 | sourcedisplay on page 42 | stack on page 43 |
| status on page 43 | step on page 44 | stop on page 45 |
| switchtarget on page 46 | system on page 46 | view on page 47 |
| wait on page 47 | watchpoint on page 48 | window on page 48 |

### alias

Creates, removes, or lists an alias for a command.

**NOTE** Aliased commands are not available from within scripts. To create a different command name or syntax, you can wrap an existing command with a Tcl proc.

```
alias [name] [definition]
```

### Shortcut

```
al
```

### Parameters

`name`

Supply the name of the alias.

`definition`

Supply the definition of the alias.

### Examples

To display all current aliases:

```
alias
```

To create a `..` alias that changes the current working directory to the parent directory:

```
alias .. cd ..
```

To remove the `..` alias:

```
alias ..
```

## bp

Sets, removes, or lists breakpoints.

```
bp
```

```
bp  func_name|machine_addr
bp  file_name line_number [column_number]
bp  func_name|brkpt_num|all OFF|enable|disable
bp  brkpt_num cond expr-elements...
```

### Shortcut

`b`

### Parameters

`func_name|machine_addr`

Supply the name or machine code address of the function on which you want to set the breakpoint.

`file_name line_number [column_number]`

Supply the name of the file, the line number, and (optionally) the column number where you want to set the breakpoint.

`func_name|brkpt_num|all OFF|enable|disable`

Supply the function name containing an existing breakpoint, the breakpoint number of an existing breakpoint, or `all`. Supply one of `OFF`, `enable`, or `disable` indicating the action you want to take on the breakpoint.

`brkpt_num cond expr-elements`

Supply the breakpoint number of an existing breakpoint, the condition to apply to the breakpoint, and the expressions you want to execute when the debugger encounters the condition.

### Examples

To display all current breakpoints:

```
bp
```

To set a breakpoint at function `fn()`:

```
bp fn
```

To set a breakpoint in file `file.cpp` at line 101, column 1.

```
bp file.cpp 101 1
```

To remove the breakpoint at function `fn()`:

```
bp fn off
```

To set a breakpoint at memory address p:10343:

```
bp p:10343
```

To remove breakpoint number 4 (use `break` to look for the number):

```
bp #4 off
```

To disable breakpoint number 4.

```
bp #4 disable
```

To set the condition for breakpoint number 4 to trigger only if x == 3:

```
bp #4 cond x == 3
```

### See also

radix on page 38

## bringtofront

Sets the floating state of the debugger console window.

```
bringtofront [on/off]
```

### Shortcut

`bri`

### Parameters

`on`

Turns floating state on for the debugger console window.

`off`

Turns the floating state off for the debugger console window.

### Examples

To toggle the floating state of the debugger console window:

```
bringtofront
```

To float the debugger console window above other IDE windows:

```
bringtofront on
```

To cause the debugger console window to return to the same layer as all other IDE windows:

```
bringtofront off
```

## cd

Changes directory.

```
cd [path]
```

### Examples

To display the current working directory:

```
cd
```

To change the current working directory to drive C:

```
cd C:
```

To change the current working directory to `D:/mw/0622/test`:

```
cd D:/mw/0622/test
```

To change the current working directory to the parent of the current working directory:

```
cd ..
```

To use a wild card to change the current working directory to `C:\Program Files`:

```
cd C:/p*s
```

To change the current working directory to `C:\notes\lib`:

```
cd C:/n*/l*
```

To change the current working directory to `C:\Acrobat3`:

```
cd c:/*3
```

### Comments

After you have entered a portion of a directory name, press Tab on the keyboard to complete the directory name automatically.

## change

Changes register or memory contents.

```
change [reg[_block]|v[arname] var|addr[_block] new_value
      [8BIT|16BIT|32BIT|64BIT]
```

### Shortcut

```
c
```

### Examples

To change register R1 to value 123 decimal:

```
change R1 123
```

To change registers R1 through R5 to decimal value 5432.

```
change R1..R5 5432
```

To change memory addresses 10 through 17 in program memory space to decimal value 3456:

```
change p:10..17 3456
```

To change the value of variable X to decimal value 16 (hexadecimal value 0x10):

```
change v X #x 0x10
```

### Comments

The `[8bit | ...]` option controls the access size for reads and writes to target memory or memory-mapped registers.

The first argument after the variable name is the format type (Table 3.3 on page 23).

**Table 3.3 Format Type Abbreviations**

| Format Type | Abbreviation | Alternate Abbreviation |
|---|---|---|
| `#Binary` | `#b` | |
| `#Boolean` | | |
| `#Char` | `#c` | |
| `#CString` | `#s` | |
| `#Default` | `#-` | |
| `#Enum` | `#e` | |
| `#Fixed` | `#i` | |
| `#Float` | `#f` | |
| `#Fract` | | |
| `#Hex` | `#h` | `#x` |
| `#PascalString` | `#p` | |
| `#Signed` | `#d` | |
| `#SignedFixed` | | |
| `#Unicode` | `#o` | `#w` |
| `#Unsigned` | `#u` | |

## cls

Clears the screen.

```
cls
```

### Shortcut

```
cl
```

## config

Configures and displays **Command Window** settings.

```
config  [project | target [TargetName]] |
        [O(nScriptError) abort | continue] |
        [C(olor) R|M|C|S|E|N txt_color [bk_color]] |
        [S(croll) lineNum]] |
        [H(exPrefix) hexadecimal_prefix] |
        [Mem(Identifier) memory_identifier] |
        [MemReadMax max_bytes] |
        [MemCache off | on] |[P(rocessor) Processorname
        [SubprocessorName]]
        [Var(iable) E(cho)|L(ocation)|F(ormat)|S(size)|T(type)
        [value]]
```

### Shortcut

```
conf
```

### Examples

To display the current config status:

```
config
```

To display the current build target:

```
config target
```

To display the current project:

config project

To change the default build target to `XXX`:

```
config target XXX
```

To abort the script if a command fails:

```
config onscripterror abort
```

To set the register display color to black, background color to white:

```
config color r $0 $0 $0 $ff $ff $ff
```

**NOTE** Refer to Table 3.4 on page 26 for a list of text color codes.

To set page-up, page-down scrolling size to hexadecimal 10 (decimal 16) lines:

```
config scroll $10
```

To display hexadecimal numbers with the prefix "0x":

```
config hexprefix 0x
```

To use "p" as the memory identifier:

```
config memidentifier p
```

To display expressions and variable names for the "evaluate" command:

```
config var echo on
```

To set default display format to decimal (see Table 3.5 on page 26):

```
config var format d
```

To disable the display of types for expressions or variables:

```
config var types off
```

To display location information for variables:

```
config var location on
```

To display size information for variables:

```
config var size on
```

To limit memory commands to 2048 (decimal) bytes, preventing a large memory read command from tying up the IDE:

```
config MemReadMax 2048
```

CodeWarrior pre-fetches chunks of memory when memory caching is on. Turning memory caching off reduces performance but provides the user with better control for memory accesses. Note that this command only works in the **Command Window**. To turn off caching of target memory:

```
config MemCache off
```

**Table 3.4  Codes for Text Color**

| Message Type | Code |
|---|---|
| command | c |
| errors | e |
| memory | m |
| normal | n |
| register | r |
| script | s |

**Table 3.5  Format Type Abbreviations**

| Format Type | Abbreviation | Alternate Abbreviation |
|---|---|---|
| `Binary` | `b` | |
| `Boolean` | | |
| `Char` | `c` | |
| `CString` | `s` | |
| `Default` | `-` | |
| `Enum` | `e` | |
| `Fixed` | `i` | |
| `Float` | `f` | |
| `Fract` | | |
| `Hex` | `h` | `x` |
| `PascalString` | `p` | |
| `Signed` | `d` | |
| `SignedFixed` | | |

Table 3.5 Format Type Abbreviations (*continued*)

| Format Type | Abbreviation | Alternate Abbreviation |
|---|---|---|
| `Unicode` | `o` | `w` |
| `Unsigned` | `u` | |

## copy

Copies memory.

```
copy addr_block addr
```

### Shortcut

```
co
```

### Examples

To copy memory addresses 00 through 1F to address 30:

```
copy p:00..1f p:30
```

To copy 10 memory locations beginning at memory address 20 to memory beginning at address 50.

```
copy p:20#10 p:50
```

### See also

radix on page 38

## debug

Starts a debugging session for a project.

```
debug [project_file(*.mcp) [number of projects]] |
     [executable_file(*.elf | *.eld)]
```

### Shortcut

```
de
```

### Examples

To debug the current default project:

```
debug
```

To open the project `des.mcp` and start debugging the default build target in it:

```
debug des.mcp
```

To start a debugging session for the project file named `8102.mcp` with three sub-projects to debug, waiting until all four projects are open before starting the debug session.

```
debug 8102.mcp 4
```

### Comments

Only use the `[number of projects]` parameter for 8102 projects.

## dir

Lists the contents of a directory.

```
dir [path|files|-d]
```

### Shortcut

```
dir
```

### Examples

```
dir
```

```
dir *.txt
```

```
dir c:/tmp
```

## disassemble

Disassembles instructions at the memory block.

```
disassemble [addr_block]
```

### Shortcut

```
di
```

### Examples

To disassemble one screenful of data, starting at the program counter:

```
disassemble
```

**NOTE** Until you run the build target again, each successive call shows the next set of instructions.

To disassemble the program memory address block 0 to 20:

```
disassemble p:0..20
```

To disassemble 16 bytes starting at memory location 0x50:

```
disassemble p:0x50#10
```

## display

Displays the contents of registers or memory

```
display [REGSET] | [OFF id] |
       [ [ON|OFF] reg_set | reg[_block] |
       [ addr[_block] [8BIT|16|BIT|32BIT|64BIT]]]
```

### Shortcut

```
d
```

### Examples

To display the default items (regset/reg/mem):

```
display
```

To list items (reg/mem/regset) on default display:

```
display on
```

To list all the available register sets on the target chip:

```
display regset
```

To display the value of register R1:

```
display R1
```

To display memory locations contents:

```
display p:00..100
```

To add all valid register sets for the current debug protocol to the default display:

```
display on ALL
```

To add "reg_set1" and "reg_set2" register sets to default display items:

```
display on <reg_set1> <reg_set2> ...
```

To add registers to default display items:

```
display on R1..R5
```

To remove "reg_set1" register set from default display items:

```
display off <reg_set1>
```

To remove memory locations from default display items:

```
display off p:230#10
```

To remove all the items from default display items:

```
display off all
```

To remove the item which ID is 2 from default display items:

```
display off #2
```

### See Also

radix on page 38

### Comments

The `[8bit | ...]` option controls the access size for reads and writes to target memory or memory-mapped registers.

Displaying a register also returns a value to Tcl. Examples

```
set myReg [display gpr0]; puts $myReg ;
```

```
set multiReg [display gpr0..gpr3]; puts $multiReg ;
```

The default unit size is 16 bits.

## evaluate

Displays C variable type or value.

```
evaluate [#formatchar|#fullformatname] [variable_Name]
```

### Shortcut

```
e
```

### Examples

To list the types for all the variables in current and global stack:

```
evaluate
```

To return the value of variable 'i':

```
evaluate i
```

To return the value of variable 'i' formatted in binary (Table 3.6 on page 32):

```
evaluate #b i
```

**Table 3.6 Format Type Abbreviations**

| Format Type | Abbreviation | Alternate Abbreviation |
|---|---|---|
| `#Binary` | `#b` | |
| `#Boolean` | | |
| `#Char` | `#c` | |
| `#CString` | `#s` | |
| `#Default` | `#-` | |
| `#Enum` | `#e` | |
| `#Fixed` | `#i` | |
| `#Float` | `#f` | |
| `#Fract` | | |
| `#Hex` | `#h` | `#x` |
| `#PascalString` | `#p` | |
| `#Signed` | `#d` | |
| `#SignedFixed` | | |
| `#Unicode` | `#o` | `#w` |
| `#Unsigned` | `#u` | |

## exit

Closes the command line window.

```
exit
```

### Shortcut

```
ex
```

## getpid

Returns the process ID of the last stopped debug process.

```
getpid
```

### Shortcut

```
ge
```

### See Also

switchtarget on page 46

## go

Start target program from the current instruction.

```
go [ALL | NOWAIT | time_period]
```

### Shortcut

```
g
```

### Comments

If run from the command window, go returns immediately.

If run from a script file, the Command Window polls for keyboard input until the target stops (for example, the target encounters a breakpoint). It will then run the next command. You may press the ESC key to stop the script if the target never stops and the Command Window continues to poll.

```
go 1
```

Stop polling the target if a breakpoint is not encountered within 1 second. The Tcl variable `still_running` is set to 1.

```
go nowait
```

If run from a script file, Tcld will execute the next script command without waiting for the target to stop.

## help

Displays help for commands.

```
help [command] | [shortcut]
```

### Shortcut

```
h
```

### Examples

Lists all the command-line debugger commands:

```
help
```

Displays help on the command `break`:

```
help break
```

Displays help on the command `break`:

```
help b
```

## history

Lists the command history.

```
history
```

### Shortcut

```
hi
```

## kill

Closes the current debug session.

```
kill [all]
```

### Shortcut

```
k
```

## log

Logs commands or a session.

```
log [OFF] [C(commands)|S(session) filename ]
```

### Shortcut

```
lo
```

### Examples

Displays currently opened log files:

```
log
```

Logs all display entries to the file `session1.log`:

```
log s session.log
```

Logs internal command contents to the file `command.log`:

```
log c command.log
```

Terminates command logging:

```
log off c
```

Terminates all logging:

```
log off
```

## make

Build the specified project or the default project if none is specified.

```
make [project file(*.mcp)]
```

### Shortcut

```
m
```

### Examples

Build the default project:

```
make
```

Build the project `test.mcp`:

```
make test.mcp
```

## next

Runs to the next source line or assembly instruction in the current frame.

```
next
```

### Shortcut

```
n
```

### Comment

The `display` command is automatically run after the `next` command finishes.

## project

Opens or closes a project file or ELF file

```
project -o[pen] file (.mcp|.elf|.elf)
```

```
project -c[lose]
```

```
project
```

### Shortcut

```
proj
```

### Examples

Open the project `des.mcp`:

```
proj -o des.mcp
```

Close the default project:

```
proj -c
```

List open projects:

```
proj
```

## pwd

Displays current working directory.

```
pwd
```

## quitIDE

Quits the CodeWarrior IDE.

```
quitIDE
```

### Shortcut

```
q
```

## radix

Changes the number base for input and memory/register displays.

```
radix [B(bin)|D(dec)|F(frc)|H(hex)|U(unsigned)]
     [reg[_block]|addr{_block] ...]
```

### Shortcut

```
r
```

### Examples

Displays the default radix currently enabled:

```
radix
```

Changes input radix to decimal:

```
radix D
```

Changes input radix to hexadecimal:

```
radix H
```

Changes the display radix for the specified registers fractional:

```
radix f r0..r7
```

Changes the display radix for the specified registers and memory blocks to decimal:

```
radix d m:0#10 r1
```

### Comments

The default value for the input and output radix is hexadecimal.

The input radix may not be changed to fractional.

Hexadecimal constants may always be specified by preceding the constant with a dollar sign ($).

Decimal constants may always be specified by preceding the constant with a grave accent (‘).

Binary constants may always be specified by preceding the constant with a percent sign (%).

## removeobj

Removes object code and binaries

```
removeobj [#a[lltargets]] [#c[ompact]] [#r[ecurse]]
    [project file(*.mcp)]
```

### Shortcut

```
rem
```

### Examples

Removes binaries for the default target for default project:

```
removeobj
```

Removes binaries for all targets for the default project:

```
removeobj #all
```

Removes binaries and compact data for the default project and all subprojects:

```
removeobj #recurse #compact
```

Removes binaries for the project `test.mcp`:

```
removeobj test.mcp
```

## reset

Attempts to reset the target system.

```
reset [h[ard] | s[oft]
```

### Shortcut

reset

### Examples

If both soft and hard reset are supported by the specific debugger plug-in, this command attempts soft reset:

```
reset
```

Perform soft reset only, if supported:

```
reset soft
```

Perform hard reset only, if supported:

```
reset hard
```

## restart

Restarts the debugging session.

```
restart
```

### Shortcut

```
re
```

### Examples

```
restart
```

This command will download the code again.

### Comments

**NOTE** For remote connections, this command causes the debugger to download code again.

If you change the debugging session memory where the program code stores the startup CRT code, the command restart will not set the PC back to the `main()` function.

## restore

Write file contents to memory

```
restore -h *.lod [addr|offset] [8bit|16bit|32bit|64bit]
restore -b *.lod addr [8bit|16bit|32bit|64bit]
```

### Shortcut

```
rest
```

### Example

Load the contents of hexfile `dat.lod` into memory:

```
restore -h dat.lod
```

Load the contents of binary file `dat.lod` into memory beginning at $20:

```
restore -b dat.lod p:$20
```

Load the contents of binary file `dat.lod` into memory with an offset of $20, relative to the address saved in `dat.lod`:

```
restore -h dat.lod $20
```

### Comments

The `[8bit | ...]` option controls the access size for reads and writes to target memory or memory-mapped registers.

### See Also

save on page 41

## save

Saves memory contents to a file.

```
save -h/-b addr_block... filename [-a/-o]
     [8bit|16bit|32bit|64bit]
```

### Shortcut

`sa`

### Examples

To save two memory blocks to `filename.lod` in hexadecimal format. If `filename.lod` exists, appends data to existing file:

```
save -h p:0..10 p:20..28 filename -a
```

To save memory blocks to `filename.lod` in binary format. If `filename.lod` exists, overwrites existing file:

```
save -b p:0..10 p:20..28 filename -o
```

### Comments

The `[8bit | ...]` option controls the access size for reads and writes to target memory or memory-mapped registers.

## sourcedisplay

Changes the source view in the front-most debugger thread window.

```
sourcedisplay [code|asm|mixed|cycle]
```

### Shortcut

so

### Examples

To cycle through available display modes:

```
sourcedisplay cycle
```

To change the view to display source code:

```
sourcedisplay code
```

To change the view to display assembly:

```
sourcedisplay asm
```

To change the view to display both source and assembly:

```
sourcedisplay mixed
```

## stack

Displays the call stack

```
stack [num_frames] [-default]
```

### Shortcut

stac

### Examples

To print the entire call stack unless limited with stack -default:

```
stack
```

To print the 6 innermost call stack levels:

```
stack 6
```

To print the 6 outer-most call stack levels:

```
stack -6
```

To limit the number of stack frames shown to the 6 innermost levels:

```
stack 6 -default
```

To remove the stack frame limit:

```
stack -default
```

## status

Displays the debug status of all active targets.

```
status
```

### Shortcut

sta

## step

Steps through the target program.

```
step [into|over|out|asm|all]
step [into|li(lines)|out|in(struction)|all]
step [nve|nxt|fwd|end|aft]
```

### Shortcut

```
st
```

### Examples

To step over a source line:

```
step
step li
step over
```

To step a single assembly instruction:

```
step asm
step in
step instruction
```

To step into a source line:

```
step into
```

To step out of a function:

```
step out
```

For supported targets, step a single assembly instruction on all cores:

```
step all
```

For supported targets, optimized code debugging step non optimized action:

```
step nve
```

For supported targets, optimized code debugging step next action:

```
step nxt
```

For supported targets, optimized code debugging step forward action:

```
step fwd
```

For supported targets, optimized code debugging step end of statement action:

```
step end
```

For supported targets, optimized code debugging step end all previous action:

```
step aft
```

### Comments

The `display` command is automatically run after a successful `step` command.

## stop

Stops the target program after the command `go`, `step out`, or `next`.

```
stop
```

### Shortcut

```
s
```

## switchtarget

During multi-core debugging, select the debug session to which the IDE sends debug commands.

```
switchtarget [pid]
```

### Shortcut

```
sw
```

### Examples

To list currently available debug sessions:

```
switchtarget
```

To select the debug session whose PID is 0:

```
switchtarget 0
```

## system

Executes a system command.

```
system [command]
```

### Shortcut

```
sy
```

### Examples

To delete any file with the extension `.tmp`:

```
system del *.tmp
```

## view

Changes view mode.

```
view [A(ssembly) [MachineAddr]] | R(egister)]
```

### Shortcut

```
v
```

### Examples

To toggle to next display mode:

```
view
```

To select assembly display mode:

```
view a
```

To select register display mode:

```
view r
```

To display assembly located from address hexadecimal 100:

```
view a $100
```

To display assembly located from address hexadecimal 100 in the specified memory space:

```
view a <memory space>:$100
```

## wait

Wait a specified time.

```
wait [milliseconds]
```

### Shortcut

```
w
```

### Examples

To wait until the user hits ESC:

```
wait
```

To wait for 2 seconds:

```
wait 2000
```

## watchpoint

Add, remove, or display a watchpoint.

```
watchpoint [variable_name|watchpoint_id OFF]
```

### Shortcut

```
wat
```

### Examples

To display the watchpoint list:

```
watchpoint
```

To add watchpoint on variable `i`:

```
watchpoint i
```

## window

Open a specific IDE debugger window

```
window [breakpoints | expressions | globals | memory |
    processes | registers | symbolics]
```

### Shortcut

```
win
```

### Examples

To open the symbolics window associated with the current debug session:

```
window
```

To open the debugger breakpoints window:

```
window breakpoints
```

To open the debugger expressions window:

```
window expressions
```

To open the debugger globals window:

```
window globals
```

To open a memory window:

```
window memory
```

To open the processes window:

```
window processes
```

To open the debugger registers window:

```
window registers
```

To open the symbolics window associated with the current debug session:

```
window symbolics
```

4

# Microsoft COM Automation

This chapter describes how to automate certain tasks performed by the CodeWarrior IDE. These tasks include: managing project files and targets, building, compiling, linking, and debugging projects, using the version control system, logging CodeWarrior messages, importing/exporting XML project files into IDE, invoking IDE menu commands, and getting the currently active document in IDE.

You may use any of the several different scripting tools (Perl, VBScript) to create automation scripts for the IDE. This chapter contains examples that use Perl and VBScript only.

This chapter has these sections:

# Viewing OLE/COM Objects

You can view the Component Object Model (COM) objects the IDE exposes and the methods you can call to work with those objects using the OLE/COM Object Viewer (Figure 4.1 on page 53). The following sub-sections describe how to work with the OLE/COM Object Viewer.

- Setting the View to Expert Mode on page 52
- Opening the Freescale Type Library on page 52
- Finding Method Details on page 55

## Setting the View to Expert Mode

The remainder of these instructions assume that you have set your Object Viewer to Expert Mode. To do so:

1. Select **View** > **Expert Mode**.

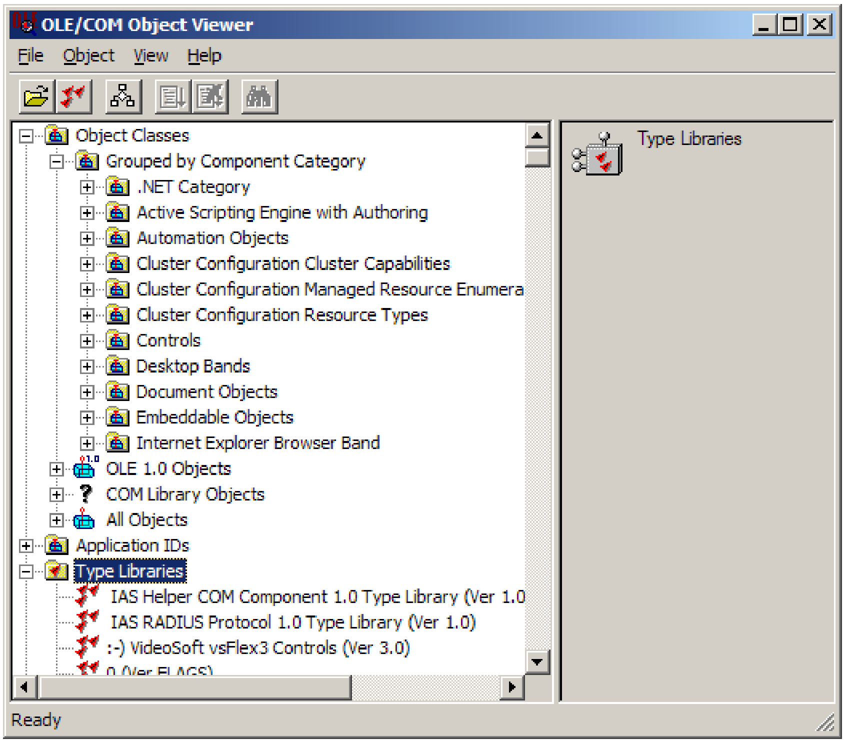
This setting provides more detail than would otherwise appear in the Object Viewer.

## Opening the Freescale Type Library

To view the interfaces and enumerations that you can use to control the IDE:

1. In the left pane, expand the **Type Libraries** list (Figure 4.1 on page 53).

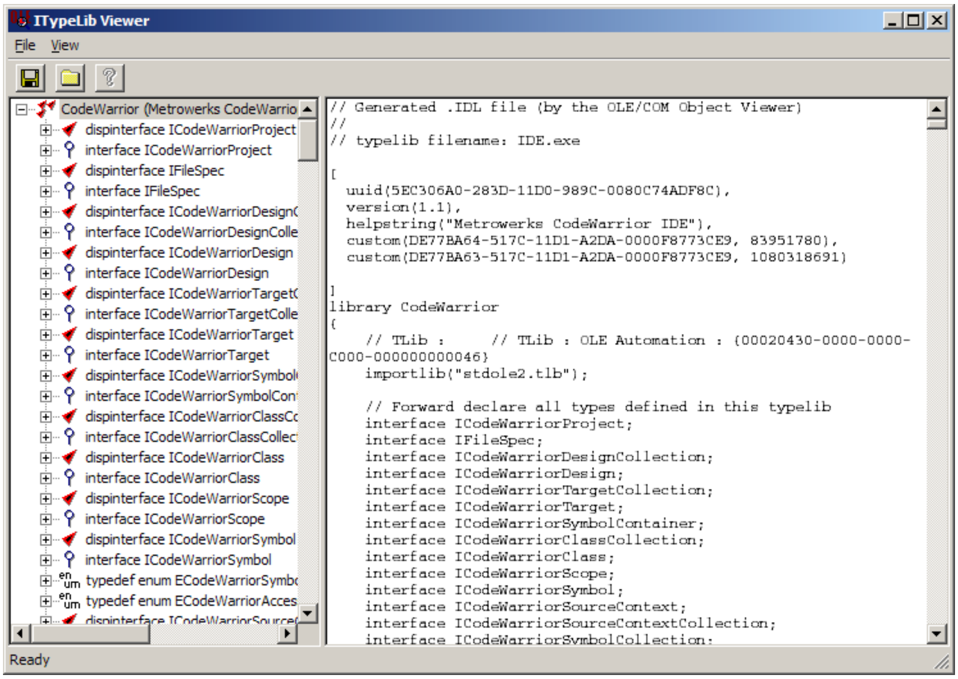
**Figure 4.1 Expand Type Libraries List**


OLE/COM Object Viewer
File Object View Help
Object Classes
Grouped by Component Category
.NET Category
Active Scripting Engine with Authoring
Automation Objects
Cluster Configuration Cluster Capabilities
Cluster Configuration Managed Resource Enumera
Cluster Configuration Resource Types
Controls
Desktop Bands
Document Objects
Embeddable Objects
Internet Explorer Browser Band
OLE 1.0 Objects
COM Library Objects
All Objects
Application IDs
Type Libraries
IAS Helper COM Component 1.0 Type Library (Ver 1.0
IAS RADIUS Protocol 1.0 Type Library (Ver 1.0)
:-) VideoSoft vsFlex3 Controls (Ver 3.0)
0 (Ver FLAGS)
Type Libraries
Ready

2. Double-click **Freescale CodeWarrior IDE** item in the **Type Libraries** tree.

   The **ITypeLib Viewer** (Figure 4.2 on page 54) appears, showing the interfaces and enumerations you can use to control the IDE.

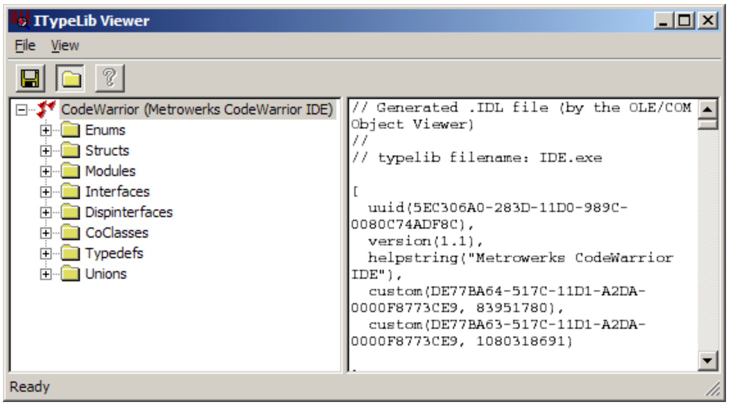
**Figure 4.2 ITypeLib Viewer**


ITypeLib Viewer
File View
CodeWarrior (Metrowerks CodeWarrio
dispinterface ICodeWarriorProject
interface ICodeWarriorProject
dispinterface IFileSpec
interface IFileSpec
dispinterface ICodeWarriorDesignC
interface ICodeWarriorDesignColle
dispinterface ICodeWarriorDesign
interface ICodeWarriorDesign
dispinterface ICodeWarriorTarget(
interface ICodeWarriorTargetColle
dispinterface ICodeWarriorTarget
interface ICodeWarriorTarget
dispinterface ICodeWarriorSymbol
interface ICodeWarriorSymbolCon
dispinterface ICodeWarriorClassCc
interface ICodeWarriorClassCollec
dispinterface ICodeWarriorClass
interface ICodeWarriorClass
dispinterface ICodeWarriorScope
interface ICodeWarriorScope
dispinterface ICodeWarriorSymbol
interface ICodeWarriorSymbol
typedef enum ECodeWarriorSymb
typedef enum ECodeWarriorAcces
Ready

```
// Generated .IDL file (by the OLE/COM Object Viewer)
//
// typelib filename: IDE.exe

[
  uuid(5EC306A0-283D-11D0-989C-0080C74ADF8C),
  version(1.1),
  helpstring("Metrowerks CodeWarrior IDE"),
  custom(DE77BA64-517C-11D1-A2DA-0000F8773CE9, 83951780),
  custom(DE77BA63-517C-11D1-A2DA-0000F8773CE9, 1080318691)

]
library CodeWarrior
{
    // TLib :     // TLib : OLE Automation : {00020430-0000-0000-
C000-000000000046}
    importlib("stdole2.tlb");

    // Forward declare all types defined in this typelib
    interface ICodeWarriorProject;
    interface IFileSpec;
    interface ICodeWarriorDesignCollection;
    interface ICodeWarriorDesign;
    interface ICodeWarriorTargetCollection;
    interface ICodeWarriorTarget;
    interface ICodeWarriorSymbolContainer;
    interface ICodeWarriorClassCollection;
    interface ICodeWarriorClass;
    interface ICodeWarriorScope;
    interface ICodeWarriorSymbol;
    interface ICodeWarriorSourceContext;
    interface ICodeWarriorSourceContextCollection;
    interface ICodeWarriorSymbolCollection;
```

3. Select **View** > **Group by type kind** in the **ITypeLib Viewer** dialog box. All the entries are grouped (Figure 4.3 on page 54).

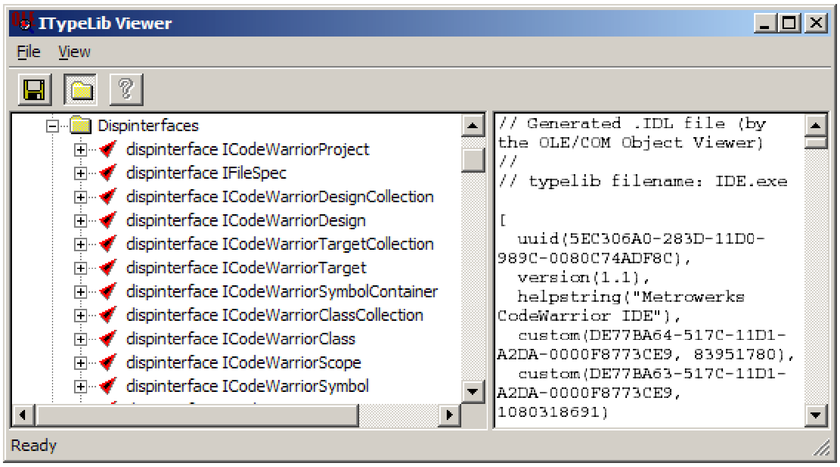
**Figure 4.3 Grouping Entries**



```
// Generated .IDL file (by the OLE/COM
Object Viewer)
//
// typelib filename: IDE.exe

[
  uuid(5EC306A0-283D-11D0-989C-
0080C74ADF8C),
  version(1.1),
  helpstring("Metrowerks CodeWarrior
IDE"),
  custom(DE77BA64-517C-11D1-A2DA-
0000F8773CE9, 83951780),
  custom(DE77BA63-517C-11D1-A2DA-
0000F8773CE9, 1080318691)
```

4. Expand the **Dispinterfaces** list (Figure 4.4 on page 55) to display the interfaces and methods you can use in a Perl script.

**Figure 4.4 Expand Dispinterfaces List**



Dispinterfaces
- dispinterface ICodeWarriorProject
- dispinterface IFileSpec
- dispinterface ICodeWarriorDesignCollection
- dispinterface ICodeWarriorDesign
- dispinterface ICodeWarriorTargetCollection
- dispinterface ICodeWarriorTarget
- dispinterface ICodeWarriorSymbolContainer
- dispinterface ICodeWarriorClassCollection
- dispinterface ICodeWarriorClass
- dispinterface ICodeWarriorScope
- dispinterface ICodeWarriorSymbol

```
// Generated .IDL file (by
the OLE/COM Object Viewer)
//
// typelib filename: IDE.exe

[
  uuid(5EC306A0-283D-11D0-
989C-0080C74ADF8C),
  version(1.1),
  helpstring("Metrowerks
CodeWarrior IDE"),
  custom(DE77BA64-517C-11D1-
A2DA-0000F8773CE9, 83951780),
  custom(DE77BA63-517C-11D1-
A2DA-0000F8773CE9,
1080318691)
```

**NOTE** Use the `Dispinterfaces` list, rather than the `Interfaces` list, when scripting in Perl. The methods in the `Dispinterfaces` list show the correct return types and parameters for Perl scripting.

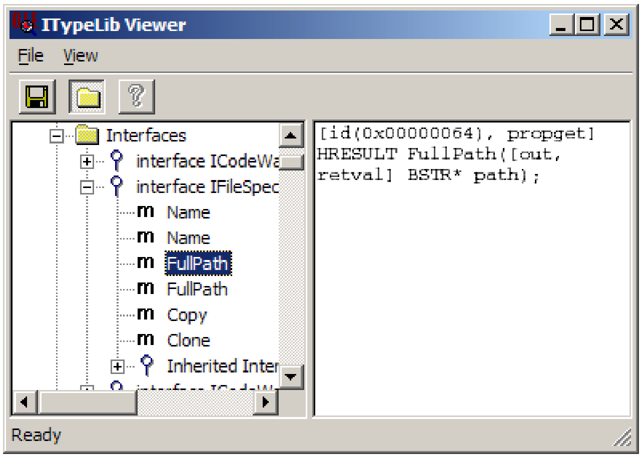
## Finding Method Details

To see the details of the methods within an interface:

1. In the ITypeLib Viewer's left pane, expand the interface you want to use from the **Interfaces** list.
2. Click the method you want to use.

   The right pane shows the definition of the selected method (Figure 4.5 on page 56).

**Figure 4.5 Select Method**



```
[id(0x00000064), propget]
HRESULT FullPath([out,
retval] BSTR* path);
```

Because the Object Viewer uses Interface Definition Language (IDL), you can see which parameters provide input and which parameters hold return values.

**NOTE** When using Perl to script CodeWarrior COM objects, remove the "I" from the beginning of each interface name. For example, use `CodeWarriorApp` rather than `ICodeWarriorApp`.

## Creating a CodeWarrior Instance

Before you can manipulate the IDE in any way, you must first create a CodeWarrior instance. The following block of code in Perl shows how to get the IDE's application object (`CodeWarriorApp`):

```
# Win32::OLE gives access to COM objects,
# including the IDE's COM # objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");
```

The following block of code in VBScript shows how to create a CodeWarrior instance:

```
set codewarrior = CreateObject("CodeWarrior.CodeWarriorApp")
```

## Managing Files in Projects

You can use Perl/VBScript scripts to add and remove files within projects.

- Adding Files to Projects on page 57
- Removing Files From Projects on page 58

### Adding Files to Projects

To add a file, you must get a reference to a project. You must then add the file to one or more targets within the project. The following script shows how to add a file to all the targets within a project.

```
# Script to add a file to all targets within a project
# Win32::OLE gives access to COM objects,
# including the IDE's COM # objects
use Win32::OLE;
# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");
# Get the command line arguments
$projecttoopen = @ARGV[0];
$filetoadd = @ARGV[1];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get the target list object
# Targets()
$targets = $project->Targets();

# Count the targets in the list
# Count()
```

```
$numtargets = $targets->Count();

# Add the file to each target
# Item (long index)
# AddFile (BSTR path,
#   BSTR groupPath)

for ($i = 0; $i < $numtargets; $i++)
{
  $targets->Item($i)->AddFile($filetoadd, "");
# end of script
```

To use this script, type:

`perl addfile.pl` *`someproject`*`.mcp` *`somefile.***`*

**NOTE** You can modify the above script to add multiple files or to read file names from an input file.

## Removing Files From Projects

To remove a file, you must get a reference to a project. You must then remove the file from one or more targets within the project. The following Perl script shows how to remove a file from all the targets within a project:

```
# Script to remove a file from all targets within a project

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0];
$filetoremove = @ARGV[1];
# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
# ECodeWarriorConvertOption convertOption,
# ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);
# Get the collection of files that match the file spec
#FindFileByName(BSTR filename)
$projectfiles =  $project->FindFileByName($filetoremove);

# Get the number of files to remove
# Count()
$filecount = $projectfiles->Count();

# Remove the files
# Item(long index)
# RemoveFile(ICodeWarriorProjectFile* projectFile)
for ($i = 0; $i < $filecount; $i++)
{
  $file = $projectfiles->Item($i);
  $project->RemoveFile($file);
}

# end of script
```

**NOTE** You can modify the above script to remove multiple files or to read file names from an input file.

## Manipulating Targets

You can manipulate the build targets in CodeWarrior IDE using the methods provided by `CodeWarriorTarget`. Some of the methods that `CodeWarriorTarget` offers are:

- AddFile on page 60
- AddSubTarget on page 60
- get_AccessPath on page 61
- get_Project on page 61
- GetSubProjects on page 61
- get_SubTargets on page 62

## AddFile

The `AddFile` method adds a file to the current target.

### Syntax

```
virtual HRESULT AddFile(
BSTR path,
BSTR groupPath,
ICodeWarriorProjectFile **projectFile);
```

where,

- `path` is the path of the file to add.
- `groupPath` is the path to the group to which to add the file.
- `projectFile` contains the address of a pointer to the project file associated with the added file.

## AddSubTarget

CodeWarrior targets can contain other CodeWarrior targets. The `AddSubTarget` method adds a target within the current target.

### Syntax

```
virtual HRESULT AddSubTarget(
ICodeWarriorTarget *target,
VARIANT_BOOL linkAgainstOutput);
```

where,

- `target` is a pointer to the target to be added.
- `linkAgainstOutput` is a link against the output of the added subtarget.

## get_AccessPath

The getAccessPath method gets the access paths for the current target.

### Syntax

```
virtual HRESULT get_AccessPaths(
ICodeWarriorAccessPaths **pval);
```

where,

- `pval` contains the address of a pointer to the access paths of the current target.

## get_Project

The `get_Project` method gets the project associated with the current target.

### Syntax

```
virtual HRESULT get_Project(
ICodeWarriorProject **project);
```

where,

- `project` contains the address of a pointer to the project associated with the current target.

## GetSubProjects

The `GetSubProjects` method gets a collection of the projects within the current target.

### Syntax

```
virtual HRESULT GetSubProjects(
ICodeWarriorSubProjectCollection **subProjectList) = 0;
```

where,

- `subProjectList` contains the address of a pointer to the a collection of subprojects.

## get_SubTargets

The `get_SubTargets` method gets the targets contained within the current target.

### Syntax

```
virtual HRESULT get_SubTargets(
ICodeWarriorSubTargetCollection **subTargetList);
```

where,

- `subTargetList` contains the address of a pointer to the collection of targets within the current target.

# Manipulating Projects

You can use Perl/VBScript scripts to manipulate projects in the IDE. You can remove the object code from a project before building it (or at any time).

## Removing Object Code From Projects

The IDE exposes separate methods for removing object code. Thus, you can remove object code at any time. However, common practice calls for removing object code before building the project.

`CodeWarriorProject` offers two methods to remove object code:

- RemoveObjectCode on page 63
- RemoveObjectCodeWithOptions on page 65

## RemoveObjectCode

The `RemoveObjectCode` method removes the object code from the specified project. This method includes an option to remove the data files created during the latest build.

The following Perl script shows an example using the `RemoveObjectCode` method:

```
# Script to remove the object code from a project

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Remove the object code
# RemoveObjectCode(ECodeWarriorWhichTargetOptions whichTarget, # 0 =
all; 1 = current
#   VARIANT_BOOL deleteDataFiles)
$project->RemoveObjectCode(0, true);

# end of script
```

The following VBScript script shows an example using the `RemoveObjectCode` method:

```
option explicit
'******Variable declaration********
dim codewarrior
dim project
dim projectname
dim targetIntf
dim count
dim projectCollection
dim targetcollection
dim result
dim showinputbox
dim objArgs
'****** Script ********
Set objArgs = Wscript.Arguments
projectname = "c:\testprojects\test1.mcp"
if objArgs.Count > 1 then
MsgBox "This Script expects only one argument, rest of the
arguments will be ignored!!"
showinputbox = false
projectname = CStr(objArgs(0))
end if
if objArgs.Count = 0 then
showinputbox = true
else
showinputbox = false
projectname = CStr(objArgs(0))
end if
if showinputbox = true then
result = InputBox("Enter the absolute path for the project to be
opened","Input", projectname, 100, 100)
If result = "" Then
projectname = "c:\testprojects\test1.mcp"
else
projectname = cstr(result)
end if
end if
'Create automation app object
set codewarrior = CreateObject("CodeWarrior.CodeWarriorApp")
MsgBox "App Created"
project = Null
'open project
set project = codewarrior.OpenProject(projectname, true, 2, 0 )
if TypeName( project ) <> "Null" then
set targetcollection = project.Targets
count = targetcollection.Count
```

```
IF ( count > 0 ) then
set targetIntf = targetcollection.Item( 0 )
targetIntf.RemoveObjectCode( true )
END IF
else
MsgBox CStr( projectname & " does not exist" )
end if
```

## RemoveObjectCodeWithOptions

The `RemoveObjectCodeWithOptions` method removes the object code from the specified project. This method includes an option to remove the data files created during the latest build and an option to remove object code from all subprojects included within the specified project.

The following Perl script shows an example using the `RemoveObjectCodeWithOptions` method:

```
# Script to remove the object code from a project and all subprojects

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);
# Remove the object code
# RemoveObjectCodeWithOptions(
# ECodeWarriorWhichTargetOptions whichTarget, # 0 = all; 1 = current
#   VARIANT_BOOL recurseSubProject,
#   VARIANT_BOOL deleteDataFiles)
$project->RemoveObjectCodeWithOptions(0, true, true);
# end of script
```

## Building Projects

`CodeWarriorProject` offers four methods to build a project:

- Build on page 66
- BuildWithOptions on page 67
- BuildAndWaitToComplete on page 68
- BuildAndWaitToCompleteWithOptions on page 71
- A Combined Example on page 73

### Build

The `Build` method builds the specified project, with no options and no error messages.

The following Perl script shows an example using the `Build` method:

```
# Script to build a project

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];
# Open the project
# OpenProject(BSTR filePath,
# VARIANT_BOOL fMakeVisible,
# ECodeWarriorConvertOption convertOption,
# ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Build the project
# Build()
$project->Build();

# end of script
```

## BuildWithOptions

The `BuildWithOptions` method builds the specified project, with the option to skip dependencies.

The following Perl script shows an example using the `BuildWithOptions` method:

```
# Script to build a project and not run after the build

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];

# Open the project
# OpenProject(BSTR filePath,
# VARIANT_BOOL fMakeVisible,
# ECodeWarriorConvertOption convertOption,
# ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);
# Build the project
# BuildWithOptions(ECodeWarriorBuildOptions options, # 0 = Normal; 1 =
Skip Dependencies
# ECodeWarriorRunMode runMode) # 0 = Don't run; 1 = Run; 2 = Run in
Debug Mode
$project->BuildWithOptions(0, 0);

# end of script
```

## BuildAndWaitToComplete

The `BuildAndWaitToComplete` method builds the specified project and waits until the build is complete to create a collection of all the messages created during the build.

The following Perl script shows an example using the `BuildAndWaitToComplete` method:

```
# Script to build a project, wait until all build messages have been
collected,
# and then print the messages

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];

# Open the project
# OpenProject(BSTR filePath,
# VARIANT_BOOL fMakeVisible,
# ECodeWarriorConvertOption convertOption,
# ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);
# Build the project
# BuildAndWaitToComplete()
$messages = $project->BuildAndWaitToComplete();

# Print the build messages
# Errors()
# ErrorCount()
# Warnings()
# WarningCount()
# Informations()
# InformationCount()
# Definitions()
# DefinitionCount()
# Item(long index)
# ErrorNumber()
# MessageText()
$errors = $messages->Errors();
$numerrors = $messages->ErrorCount();
$warnings = $messages->Warnings();
$numwarnings = $messages->WarningCount();
$informations = $messages->Informations();
$numinformations = $messages->InformationCount();
$definitions = $messages->Definitions();
$numdefinitions = $messages->DefinitionCount();

print ("Number of Errors: $numerrors\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numerrors; $i++)
{
  $errortoprint = $errors->Item($i);
  $errornum = $errortoprint->ErrorNumber();
  $stringtoprint = $errortoprint->MessageText();
  print("$errornum: $stringtoprint\n");
}

print ("\nNumber of Warnings: $numwarnings\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numwarnings; $i++)
{
  $warningtoprint = $warnings->Item($i);
  $Warningnum = $warningtoprint->ErrorNumber();
  $stringtoprint = $warningtoprint->MessageText();
  print("$warningnum: $stringtoprint\n");
}

print ("\nNumber of Informations: $numinformations\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numinformations; $i++)
{
  $informationtoprint = $informations->Item($i);
  $informationnum = $informationtoprint->ErrorNumber();
  $stringtoprint = $informationtoprint->MessageText();
  print("$informationnum: $stringtoprint\n");
}
```

```
print ("\nNumber of Definitions: $numdefinitions\n");
print ("--------------------------------\n");
for ($i = 0; $i < $numdefinitions; $i++)
{
  $definitiontoprint = $definitions->Item($i);
  $definitionnum = $definitiontoprint->ErrorNumber();
  $stringtoprint = $definitiontoprint->MessageText();
  print("$definitionnum: $stringtoprint\n");
}

# end of script
```

The following VBScript script shows an example using the `BuildAndWaitToComplete` method:

```
option explicit
'*******Variable declaration
dim codewarrior
dim project
dim projectname
dim targetIntf
dim count
dim projectCollection
dim targetcollection
dim result
dim showinputbox
dim objArgs
dim buildErrors
'****** Script ********
Set objArgs = Wscript.Arguments
projectname = "c:\temp\none\none.mcp

if objArgs.Count > 1 then

MsgBox "This Script expects only one argument, rest of the
arguments will be ignored!!"
showinputbox = false
projectname = CStr(objArgs(0))
end if
if objArgs.Count = 0 then
showinputbox = true
else
showinputbox = false
```

```
projectname = CStr(objArgs(0))
end if
if showinputbox = true then
result = InputBox("Enter the absolute path for the project to be
opened","Input", projectname, 100, 100)
If result = "" Then
projectname = "c:\testprojects\test1.mcp"
else
projectname = cstr(result)
end if
end if
'Create automation app object
set codewarrior = CreateObject("CodeWarrior.CodeWarriorApp")
MsgBox "App Created"
project = Null
'open project
set project = codewarrior.OpenProject(projectname, true, 2, 0 )
if TypeName( project ) <> "Null" then
project.BuildAndWaitToComplete
else
MsgBox CStr( projectname & " does not exist" )
end if
project.close
```

## BuildAndWaitToCompleteWithOptions

The `BuildAndWaitToCompleteWithOptions` method builds the specified project and waits until the build is complete to create a collection of all the messages created during the build. It offers the option to skip dependencies.

The following Perl script shows an example using the `BuildAndWaitToCompleteWithOptions` method:

```
# Script to build a project, wait until all build messages have been
collected,
# and then print the messages

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;
```

```
# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Build the project
# BuildAndWaitToCompleteWithOptions(
#   ECodeWarriorBuildOptions options) # 0 = Normal; 1 = Skip
Dependencies
$messages = $project->BuildAndWaitToCompleteWithOptions(0);

# Print the build messages
# Errors()
# ErrorCount()
# Warnings()
# WarningCount()
# Informations()
# InformationCount()
# Definitions()
# DefinitionCount()
# Item(long index)
# ErrorNumber()
# MessageText()
$errors = $messages->Errors();
$numerrors = $messages->ErrorCount();
$warnings = $messages->Warnings();
$numwarnings = $messages->WarningCount();
$informations = $messages->Informations();
$numinformations = $messages->InformationCount();
$definitions = $messages->Definitions();
$numdefinitions = $messages->DefinitionCount();

print ("Number of Errors: $numerrors\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numerrors; $i++)
{
  $errortoprint = $errors->Item($i);
  $errornum = $errortoprint->ErrorNumber();
  $stringtoprint = $errortoprint->MessageText();
  print("$errornum: $stringtoprint\n");
}

print ("\nNumber of Warnings: $numwarnings\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numwarnings; $i++)
{
  $warningtoprint = $warnings->Item($i);
  $Warningnum = $warningtoprint->ErrorNumber();
  $stringtoprint = $warningtoprint->MessageText();
  print("$warningnum: $stringtoprint\n");
}

print ("\nNumber of Informations: $numinformations\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numinformations; $i++)
{
  $informationtoprint = $informations->Item($i);
  $informationnum = $informationtoprint->ErrorNumber();
  $stringtoprint = $informationtoprint->MessageText();
  print("$informationnum: $stringtoprint\n");
}

print ("\nNumber of Definitions: $numdefinitions\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numdefinitions; $i++)
{
  $definitiontoprint = $definitions->Item($i);
  $definitionnum = $definitiontoprint->ErrorNumber();
  $stringtoprint = $definitiontoprint->MessageText();
  print("$definitionnum: $stringtoprint\n");
}

# end of script
```

## A Combined Example

Build scripts often remove object code from a project and then build the project. The following example in Perl uses `RemoveObjectCodeWithOptions` and `BuildAndWaitToComplete` to perform those tasks:

```
# Script to remove all object code, build a project, wait until
# all build messages have been collected, and print the messages

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line argument
$projecttoopen = @ARGV[0];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Remove the object code
# RemoveObjectCodeWithOptions(
# ECodeWarriorWhichTargetOptions whichTarget, # 0 = all; 1 = current
#   VARIANT_BOOL recurseSubProject,
#   VARIANT_BOOL deleteDataFiles)
$project->RemoveObjectCodeWithOptions(0, true, true);

# Build the project
# BuildAndWaitToComplete()
$messages = $project->BuildAndWaitToComplete();

# Print the build messages
# Errors()
# ErrorCount()
# Warnings()
# WarningCount()
# Informations()
# InformationCount()
# Definitions()
# DefinitionCount()
# Item(long index)
# ErrorNumber()
# MessageText()
$errors = $messages->Errors();
$numerrors = $messages->ErrorCount();
$warnings = $messages->Warnings();
$numwarnings = $messages->WarningCount();
```

```
$informations = $messages->Informations();
$numinformations = $messages->InformationCount();
$definitions = $messages->Definitions();
$numdefinitions = $messages->DefinitionCount();

print ("Number of Errors: $numerrors\n");
print ("---------------------------------\n");
for ($i = 0; $i < $numerrors; $i++)
{
  $errortoprint = $errors->Item($i);
  $errornum = $errortoprint->ErrorNumber();
  $stringtoprint = $errortoprint->MessageText();
  print("$errornum: $stringtoprint\n");
}

print ("\nNumber of Warnings: $numwarnings\n");
print ("---------------------------------\n");
for ($i = 0; $i < $numwarnings; $i++)
{
  $warningtoprint = $warnings->Item($i);
  $Warningnum = $warningtoprint->ErrorNumber();
  $stringtoprint = $warningtoprint->MessageText();
  print("$warningnum: $stringtoprint\n");
}

print ("\nNumber of Informations: $numinformations\n");
print ("---------------------------------\n");
for ($i = 0; $i < $numinformations; $i++)
{
  $informationtoprint = $informations->Item($i);
  $informationnum = $informationtoprint->ErrorNumber();
  $stringtoprint = $informationtoprint->MessageText();
  print("$informationnum: $stringtoprint\n");
}

print ("\nNumber of Definitions: $numdefinitions\n");
print ("---------------------------------\n");
for ($i = 0; $i < $numdefinitions; $i++)
{
  $definitiontoprint = $definitions->Item($i);
  $definitionnum = $definitiontoprint->ErrorNumber();
  $stringtoprint = $definitiontoprint->MessageText();
  print("$definitionnum: $stringtoprint\n");
}

# end of script
```

## Compiling Projects

You can compile collections of files within a project or target. The following sections explain how to do:

- Compiling From Projects on page 76
- Compiling From Build Targets on page 77

### Compiling From Projects

`CodeWarriorProject` offers one method for compiling collections of files (including collection that consist of one file):

#### CompileFilesWithChoice

The `CompileFilesWithChoice` method performs one of the following actions on the specified collection of files:

- Check Syntax
- Preprocess
- Precompile
- Compile
- Disassemble

Because `CompileFilesWithChoice` associates with the project, it compiles the file for all targets. See "Compiling From Build Targets" on page 77 for how to compile files for a single target.

The following script shows how to use `CompileFilesWithChoice` to compile an individual file within a project:

```
# Script to compile a file within a project

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");
```

```
# Get the command line arguments
$projecttoopen = @ARGV[0];
$filetocompile = @ARGV[1];
# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get a file collection (required by the compiling method)
$filecoll = $project->FindFileByName($filetocompile);


#Compile the file
# CompileFilesWithChoice(
#   ICodeWarriorProjectFileCollection* collection,
#   ECodeWarriorCompileChoice compileChoice);
# ECodeWarriorCompileChoice
#   0 = Check Syntax
#   1 = Preprocess
#   2 = Precompile
#   3 = Compile
#   4 = Disassemble
$project->CompileFilesWithChoice($filecoll, 3);

# Note: Ignoring the return value

# end of script
```

This example script compiles a single file, but you can modify it to compile a number of files or to read filenames from an input file.

## Compiling From Build Targets

The `CodeWarriorTarget` method offers three methods for compiling collections of files (including collection that consist of one file):

- CompileFiles on page 78
- CompileFilesAndWaitToComplete on page 79
- CompileFilesWithChoice on page 81

## CompileFiles

The `CompileFiles` method compiles the specified collection of files within the target.

The following Perl script shows how to use `CompileFiles` to compile an individual file within a target:

```
# Script to compile a file within a target

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0];
$targettoopen = @ARGV[1];
$filetocompile = @ARGV[2];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get the target
$targettouse = $project->FindTarget($targettoopen);

# Get the project file collection
#   containing the file to compile
$filecoll = $project->FindFileByName($filetocompile);

# Compile the file
# CompileFiles(ICodeWarriorProjectFileCollection* collection);
$targettouse->CompileFiles($filecoll);

# end of script
```

This example script compiles a single file, but you can modify it to compile a number of files or to read filenames from an input file.

## CompileFilesAndWaitToComplete

The `CompileFilesAndWaitToComplete` method compiles the specified collection of files within the target. `CompileFilesAndWaitToComplete` generates messages, which your script can print or save.

The following Perl script shows how to use `CompileFilesAndWaitToComplete` to compile an individual file within a target:

```
# Script to compile a file within a target,
# gather the resulting messages,
# and print the messages

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0];
$targettoopen = @ARGV[1];
$filetocompile = @ARGV[2];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get the target
$targettouse = $project->FindTarget($targettoopen);

# Get the project file collection
#   containing the file to compile
$filecoll = $project->FindFileByName($filetocompile);
# Compile the file and create the messages
# CompileFilesAndWaitToComplete(ICodeWarriorProjectFileCollection*
collection);
$messages = $targettouse->CompileFilesAndWaitToComplete($filecoll);
```

```
# Print the messages
# Errors()
# ErrorCount()
# Warnings()
# WarningCount()
# Informations()
# InformationCount()
# Definitions()
# DefinitionCount()
# Item(long index)
# ErrorNumber()
# MessageText()
$errors = $messages->Errors();
$numerrors = $messages->ErrorCount();
$warnings = $messages->Warnings();
$numwarnings = $messages->WarningCount();
$informations = $messages->Informations();
$numinformations = $messages->InformationCount();
$definitions = $messages->Definitions();
$numdefinitions = $messages->DefinitionCount();

print ("Number of Errors: $numerrors\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numerrors; $i++)
{
  $errortoprint = $errors->Item($i);
  $errornum = $errortoprint->ErrorNumber();
  $stringtoprint = $errortoprint->MessageText();
  print("$errornum: $stringtoprint\n");
}

print ("\nNumber of Warnings: $numwarnings\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numwarnings; $i++)
{
  $warningtoprint = $warnings->Item($i);
  $Warningnum = $warningtoprint->ErrorNumber();
  $stringtoprint = $warningtoprint->MessageText();
  print("$warningnum: $stringtoprint\n");
}

print ("\nNumber of Informations: $numinformations\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numinformations; $i++)
{
  $informationtoprint = $informations->Item($i);
  $informationnum = $informationtoprint->ErrorNumber();
  $stringtoprint = $informationtoprint->MessageText();
  print("$informationnum: $stringtoprint\n");
}

print ("\nNumber of Definitions: $numdefinitions\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numdefinitions; $i++)
{
  $definitiontoprint = $definitions->Item($i);
  $definitionnum = $definitiontoprint->ErrorNumber();
  $stringtoprint = $definitiontoprint->MessageText();
  print("$definitionnum: $stringtoprint\n");
}

# end of script
```

This example script compiles a single file, but you can modify it to compile a number of files or to read filenames from an input file.

## CompileFilesWithChoice

The `CompileFilesWithChoice` method performs one of the following actions on the specified collection of files:

- Check Syntax
- Preprocess
- Precompile
- Compile
- Disassemble

The following Perl script shows how to use `CompileFilesWithChoice` to compile an individual file within a target:

```
# Script to perform one of a number of possible actions
# on a file within a target

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0];
$targettoopen = @ARGV[1];
$filetocompile = @ARGV[2];
$action = @ARGV[3];

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get the target
$targettouse = $project->FindTarget($targettoopen);

# Get the project file collection
#   containing the file to compile
$filecoll = $project->FindFileByName($filetocompile);

# Compile the file
# CompileFilesWithChoice(
#   ICodeWarriorProjectFileCollection* collection,
#   ECodeWarriorCompileChoice compileChoice);
# ECodeWarriorCompileChoice:
#   0 = Check Syntax
#   1 = Preprocess
#   2 = Precompile
#   3 = Compile
#   4 = Disassemble
$targettouse->CompileFilesWithChoice($filecoll, $action);

# Note: Ignoring the return value

# end of script
```

This example script compiles a single file, but you can modify it to compile a number of files or to read filenames from an input file.

## Linking Projects

`CodeWarriorTarget` lets you obtain the linker name and specify how to link against specific files in targets.

### Obtaining the Linker Name

The COM Application Programming Interface (API) exposes a method that lets you obtain the name of the current linker plug-in. To do so, use:

- GetLinkerName on page 83

#### GetLinkerName

The `GetLinkerName` method obtains the name of the linker for a target.

The following Perl script shows how to use `GetLinkerName` to obtain the name of the linker for a target:

```
# Script to get the name of the current linker

# Win32::OLE gives access to COM objects,
# including the IDE’s COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0];
$targettoopen = @ARGV[1];
# Open the project
# OpenProject(BSTR filePath,
# VARIANT_BOOL fMakeVisible,
# ECodeWarriorConvertOption convertOption,
# ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);
```

```
# Get the target
# FindTarget(BSTR Name)
$target = $project->FindTarget($targettoopen);

# Get the linker name
# GetLinkerName()
$linkername = $target->GetLinkerName();

# Print the linker name
print("Linker for $targettoopen: $linkername\n");

# end of script
```

## Linking Against Sub-Targets

The COM API exposes a method that lets you specify how to link against subtargets. To do so, use:

- `LinkAgainstSubTarget on page 84`

### LinkAgainstSubTarget

The `LinkAgainstSubTarget` method set whether to link against a specified subtarget within a target.

The following Perl script shows how to use `LinkAgainstSubTarget` to set whether to link against a specified subtarget within a target:

```
# Script to set whether to link against a particular subtarget

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");
# Get the command line arguments
$projecttoopen = @ARGV[0]; # Use the full path
```

```
$targettoopen = @ARGV[1]; # Use the target name
$subtargettoopen = @ARGV[2]; # Use the subtarget name
$linkornot = @ARGV[3];  # Use true or false

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get the target
# FindTarget(BSTR Name)
$target = $project->FindTarget($targettoopen);

# Get the subtarget and set whether to link against it
# SubTargets()
# Count()
# Item(long index)
# Target()
# Name()
# LinkAgainstSubTarget(
#   ICodeWarriorSubTarget* Target,
#   VARIANT_BOOL val);
$subtargs = $target->SubTargets();
$numsubtargs = $subtargs->Count();

for($i = 0; $i < $numsubtargs; $i++)
{
  if ($subtargs->Item($i)->Target()->Name() eq $subtargettoopen)
  {
    $subtargettouse = $subtargs->Item($i);
    $target->LinkAgainstSubTarget($subtargettouse, $linkornot);
    exit;
  }
}
```

This example script sets whether to link against a single target, but you can modify it to link against a number of subtargets or to read subtarget names from an input file.

## Linking Against Sub-Projects

The COM API exposes a method that lets you specify whether to link against subprojecttargets (that is, targets within subprojects). To do so, use:

- LinkAgainstSubProjectTarget on page 86.

## LinkAgainstSubProjectTarget

The `LinkAgainstSubProjectTarget` method sets whether to link against a specified subprojecttarget within a target.

The following Perl script shows how to use `LinkAgainstSubProjectTarget` to set whether to link against a specified subprojecttarget within a target:

```
# Script to set whether to link against a particular subtarget

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0]; # Use the full path
$targettoopen = @ARGV[1]; # Use the target name
$subprojecttoopen = @ARGV[2]; # Use the subprojectname
$subprojecttargettoopen = @ARGV[3]; # Use the subprojecttarget name
$linkornot = @ARGV[4];  # Use true or false
# Open the project
# OpenProject(BSTR filePath,
# VARIANT_BOOL fMakeVisible,
# ECodeWarriorConvertOption convertOption,
# ECodeWarriorRevertPanelOption revertOption)
$project = $CW->OpenProject($projecttoopen, true, 0, 0);

# Get the target
# FindTarget(BSTR Name)
$target = $project->FindTarget($targettoopen);
```

```
# Get the subproject, get the subprojecttarget,
# and set whether to link against it
# GetSubProjects()
# Count()
# Item(long index)
# Targets()
# Name()
# LinkAgainstSubProjectTarget(
#   ICodeWarriorSubProjectTarget* Target,
#   VARIANT_BOOL val);
$subpjcts = $target->GetSubProjects();
$numsubpjcts = $subpjcts->Count();

for($i = 0; $i < $numsubpjcts; $i++)
{
  if ($subpjcts->Item($i)->Name() eq $subprojecttoopen)
  {
    $subpjcttouse = $subpjcts->Item($i);
    $subpjctargets = $subpjcttouse->Targets();
    $numsubpjcttargets = $subpjcttargets->Count();
    for($j = 0; $j < $numsubpjcttargets; $j++)
    {
      if ($subpjcttgts->Item($j)->Name() eq $subprjttargettoopen)
      {
        $target->LinkAgainstSubProjectTarget(subpjcttargets-
>Item($j));
        exit; # stop at the first match
      }
    }
  }
}
```

This example script sets whether to link against a single subproject within a target, but you can modify it to link against a number of sub projects within a target or to read subproject target names from an input file.

## Generating Debugger Output

Using the COM API to debug, actually tells the IDE to build the target and create the debugging output. You can then capture the output for display or saving.

## Debugging a Target

`CodeWarriorTarget` offers a single method for debugging a target:

- Debug on page 88

## Debug

`Debug` starts a debugging session for a target.

To use the `Debug` method, you must first use the `SetupDebugging` method, as shown in the sample script.

The following Perl script shows how to use `Debug`:

```
# Script to debug a target and print the resulting messages

# Win32::OLE gives access to COM objects,
# including the IDE's COM objects
use Win32::OLE;

# Create an instance of CodeWarrior
$CW = Win32::OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0]; # Use the full path
$targettodebug = @ARGV[1]; # Use the target name

# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)

$project = $CW->OpenProject($projecttoopen, true, 0, 0);
# Get the target
# FindTarget(BSTR Name)
$target = $project->FindTarget($targettodebug);

# Enable debugging for this target
# SetupDebugging(VARIANT_BOOL inTurnOn)
$target->SetupDebugging(true);

# Start debugging
# Debug()
$messages = $target->Debug();

# Print the messages
# Errors()
# ErrorCount()
# Warnings()
# WarningCount()
# Informations()
# InformationCount()
# Definitions()
# DefinitionCount()
# Item(long index)
# ErrorNumber()
# MessageText()
$errors = $messages->Errors();
$numerrors = $messages->ErrorCount();
$warnings = $messages->Warnings();
$numwarnings = $messages->WarningCount();
$informations = $messages->Informations();
$numinformations = $messages->InformationCount();
$definitions = $messages->Definitions();
$numdefinitions = $messages->DefinitionCount();

print ("Number of Errors: $numerrors\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numerrors; $i++)
{
  $errortoprint = $errors->Item($i);
  $errornum = $errortoprint->ErrorNumber();
  $stringtoprint = $errortoprint->MessageText();
  print("$errornum: $stringtoprint\n");
}

print ("\nNumber of Warnings: $numwarnings\n");
print ("----------------------------------\n");
for ($i = 0; $i < $numwarnings; $i++)
{
  $warningtoprint = $warnings->Item($i);
  $Warningnum = $warningtoprint->ErrorNumber();
  $stringtoprint = $warningtoprint->MessageText();
  print("$warningnum: $stringtoprint\n");
}
```

```
print ("\nNumber of Informations: $numinformations\n");
print ("---------------------------------\n");
for ($i = 0; $i < $numinformations; $i++)
{
  $informationtoprint = $informations->Item($i);
  $informationnum = $informationtoprint->ErrorNumber();
  $stringtoprint = $informationtoprint->MessageText();
  print("$informationnum: $stringtoprint\n");
}

print ("\nNumber of Definitions: $numdefinitions\n");
print ("---------------------------------\n");
for ($i = 0; $i < $numdefinitions; $i++)
{
  $definitiontoprint = $definitions->Item($i);
  $definitionnum = $definitiontoprint->ErrorNumber();
  $stringtoprint = $definitiontoprint->MessageText();
  print("$definitionnum: $stringtoprint\n");
}

# end of script
```

This example script generates debugging information for a single target, but you could modify it to work for multiple targets or to read targets from an input file. You could also write the resulting output to text files.

## Displaying IDE Messages

The COM API lets you log CodeWarrior IDE messages on your screen.

### Logging IDE Output

`CodeWarriorBuildMessages` and `CodeWarriorMessages` offers the following methods for logging IDE messages:

- Errors
- ErrorCount
- MessageText

The following Perl script shows how to use the above mentioned methods:

```
# compile and get list of messages (CodeWarriorBuildMessages)
   my $messages = $target->BuildAndWaitToComplete();

   # if messages undefined, maybe the build hung
   if ( !defined($messages) )
   {
       print LOG ("Messages undefined! perhaps the IDE hung.\n");
       exit(1);
   }

   # report any errors
   if ($messages->ErrorCount > 0)
  {

       print LOG ("---------------------------------------------\n");
       print LOG ($messages->ErrorCount . " errors on build:\n");

       # print out the version of CodeWarrior actually used
       $toolpath = $CW->FullName();
       print LOG ("buildtool is $toolpath\n");

       # CodeWarriorMessageCollection $errors
       my $errors = $messages->Errors();

       for (my $i = 0; $i < $errors->Count(); $i++)
       {
          # CodeWarriorMessage
          my $m = $errors->Item($i);
          print LOG ( "\n" );
         print LOG (substr($m->FileSpec->FullPath, $dirLen+1) . "\n");
          if ( defined($m->projectFile) )
          { print LOG (substr($m->projectFile->Name, $dirLen+1) .
          "\n"); }
          else { print LOG ("message project file not defined!\n"); }
          if (defined($m->Target))
          { print LOG ($m->Target()->Name() . "\n"); }
          else { print LOG ("message target not defined!\n"); }
          print LOG ( "\n" . $m->MessageText() . "\n" );
       }

       print LOG ("---------------------------------------------\n");
   }
```

## Using Version Control System

The COM API lets you check files into and out of a version control system. To use the version control methods, you must have set the various version control settings in the IDE, either for the current project or globally.

`CodeWarriorProject` offers the [`VersionControl on page 92`] method for accessing a version control system.

### VersionControl

The following Perl script shows how to use `VersionControl`:

```
# Script to perform VCS operations on files
# of a specified type within a project

# Win32OLE gives access to COM objects,
# including the IDE¼s COM objects
use Win32OLE;

# Create an instance of CodeWarrior
$CW = Win32OLE->new("CodeWarrior.CodeWarriorApp");

# Get the command line arguments
$projecttoopen = @ARGV[0]; # Use the full path
$filename = @ARGV[1]; # Use the file name or wildcards
                      # (such as "*.c")
$inorout = @ARGV[2]; # Use "checkin" or "checkout"
# Open the project
# OpenProject(BSTR filePath,
#   VARIANT_BOOL fMakeVisible,
#   ECodeWarriorConvertOption convertOption,
#   ECodeWarriorRevertPanelOption revertOption)

$project = $CW->OpenProject($projecttoopen, true, 0, 0);
```

```
# Get the filecollection object
# FindFileByName(BSTR fileName)
$filecoll = $project->FindFileByName($filename);

# Get the number of files
# Count()
$numfiles = $filecoll->Count();

# Get the version control client
# VersionControl()
$vcc = $project->VersionControl();

# Connect to the version control database
# Connect()
# IsConnected()
if (!$vcc->IsConnected())
{
  $vcc->Connect();
}

# Perform the VCS operation
if ($inorout eq "checkin")
{
  for($i = 0; $i < $numfiles; $i++)
  {
    $filetocheck = $filecoll->Item($i);
    $state = $filetocheck->VCSState()->CKIDState();
    $thisfilename = $filetocheck->Name();
    if($state == 0)
    {
      print("

    }
    elsif ($state == 1)
    {
      print("

    }
    elsif ($state == 2)
    {
      print("

    }
    elsif ($state == 3)
    {
      print("
      $filetocheck->Checkin();
    }
    elsif ($state == 4)
    {
      print("

    }
  }
}
else
{
  for($i = 0; $i < $numfiles; $i++)
  {
    $filetocheck = $filecoll->Item($i);
    $state = $filetocheck->VCSState()->CKIDState();
    $thisfilename = $filetocheck->Name();
    if($state == 0)
    {
      print("

    }
    elsif ($state == 1)
    {
      print("

    }
    elsif ($state == 2)
    {
      print("

      $filetocheck->Checkout();
    }
    elsif ($state == 3)
    {
      print("

    }
    elsif ($state == 4)
    {
      print("

      $filetocheck->Checkout();
    }
  }
}

# Disconnect from the version control database
# Disconnect()
# IsConnected()
if ($vcc->IsConnected())
{
  $vcc->Disconnect();
}

# end of script
```

## Importing and Exporting Project XML Files

The COM API lets you export or import an XML project file into the CodeWarrior IDE. `CodeWarriorApp` offers the ImportProject on page 95 method to import an XML project file. `CodeWarriorProject` offers the Export on page 96 method to create an XML file containing the project file details.

### ImportProject

Use the `ImportProject` method to import an XML project file into the CodeWarrior IDE, specifying the full path to the import file.

#### Syntax

```
virtual HRESULT ImportProject(
BSTR textFilePath,
BSTR projectFilePath,
VARIANT_BOOL fMakeVisible,
ICodeWarriorProject **pval) = 0;
```

where,

- `textFilePath` is the full path to the XML file you are importing.
- `projectFilePath` is the full path to the new project file. This file must not exist. It is created by CodeWarrior.
- `pval` contains the address of a pointer to the resulting project.

## Export

Use the `Export` method to create an XML file containing the details of the project file.

### Syntax

```
virtual HRESULT ImportProject(
BSTR textFilePath,
BSTR projectFilePath,
VARIANT_BOOL fMakeVisible,
ICodeWarriorProject **pval) = 0;
```

where,

- `textFilePath` is the full path to the XML file you are importing.
- `projectFilePath` is the full path to the new project file. This file must not exist. It is created by CodeWarrior.
- `pval` contains the address of a pointer to the resulting project.

The following VBScript script shows how to use the `Export` method:

```
' This script was created as an exercise for the students in
' the Scripting CodeWarrior course available from Freescale.
' This script creates an application object, gets the default
' project. It then Exports the entire project in XML format
' inorder to read the XML and find all Groups in the File View.
' All Groups are displayed along with the files which are
' contained within each Group.
option explicit
dim CW 'ICodeWarrior
dim project'default project
dim textDocument'text document object to hold report
dim textEngine'the object for dealing with text
dim eol 'end-of-line character for formatting
dim result'returned values
dim projectName 'name of default project
dim FileSpecFor_xml 'XML file name
dim FSO 'IFileSystem
dim TS 'ITextStream
dim fileLine 'text line read from xml file
dim i 'loop; math
dim j 'math
dim GroupName 'group name
dim File 'file within current group
dim GroupLevel 'group nesting level
dim GroupCount 'number of groups
dim foundit 'loop flag
eol = chr(13)'set end of line character
' create an instance of CodeWarrior
set CW = CreateObject("CodeWarrior.CodeWarriorApp")
'create text document and get engine
set textDocument = CW.OpenUntitledTextDocument()
set textEngine = textDocument.TextEngine
'get the default project
set project = CW.DefaultProject
'do some error control here
if TypeName(project) = "Nothing" then
textEngine.InsertText("Script operates on default project."
&eol)
textEngine.InsertText("There must be at least one open
project." &eol)
else ' valid project
' Export the project as XML, open that XML file,
' read down to the <GROUPLIST> line and then parse
' for:
' <GROUP><NAME>GroupName</NAME> = beginning of group
' with name of group;
' </GROUP> = end of group; and
' <PATH>Filename</PATH> = file within current
' group.
projectName = project.Name
textEngine.InsertText("Structure of project: " &projectName
&eol)
```

```
textEngine.InsertText("============================" &eol)
' *** get fullpath filespec for project and append ".xml"
FileSpecFor_xml = project.FileSpec.FullPath & ".xml"
' *** export entire project in XML format
project.Export(FileSpecFor_xml)
' *** Xml file exists, try to open it
set FSO = CreateObject("Scripting.FileSystemObject")
if ( NOT FSO.FileExists( FileSpecFor_xml ) ) then
textEngine.InsertText("**Couldn''t open " &FileSpecFor_xml
&eol)
end if
set TS = FSO.OpenTextFile( FileSpecFor_xml )
GroupLevel = 0 ' group Nesting Level
GroupCount = 0 ' number of Groups
' *** find beginning of <GROUPLIST> in xml file
foundit = False
do
fileLine = TS.Readline
if( instr( fileLine, "<GROUPLIST>") <> 0 ) then
foundit = True
end if
loop until( foundit OR TS.AtEndOfStream )
do while( NOT TS.AtEndOfStream ) ' if Not EOF
fileLine = TS.Readline
if( instr( fileLine, "<GROUP>" ) <> 0) then ' if group
i = 6 + instr( fileLine, "<NAME>" )
j = instr( fileLine, "</NAME>" )
GroupName = mid( fileLine, i, j-i) ' extract
group name
for i=0 to GroupLevel
textEngine.InsertText("|--")
next
textEngine.InsertText("Group: " &GroupName &eol)
GroupLevel = GroupLevel + 1
GroupCount = GroupCount + 1
elseif( instr(fileLine, "</GROUP>") <> 0 ) then ' if end of
group
if(GroupLevel>0) then
GroupLevel = GroupLevel - 1 ' decr. level
if non-zero
end if
elseif( instr(fileLine, "<PATH>") <> 0) then ' if file
i = 6 + instr( fileLine, "<PATH>")
j = instr( fileLine, "</PATH>")
File = mid( fileLine, i, j-i) ' extract
```

```
file name
for i=0 to GroupLevel
textEngine.InsertText("|--")
next
textEngine.InsertText("File: " &File &eol)
end if
loop ' while not EOF
TS.close
textEngine.InsertText(" " &eol)
textEngine.InsertText("..Total number of Groups: "
&GroupCount &eol)
end if ' valid project
```

## Invoking IDE Menu Commands

The `CodeWarriorApp` offers the `DoCommand` method to invoke a menu command in the CodeWarrior IDE.

**NOTE** For information on command ID's, see the `CodeWarriorCommandNumbers.h` file located at the following location in your CodeWarrior installation directory: `\(CodeWarrior SDK)\COM`.

The following VBScript script shows how to use the `DoCommand` method:

```
'This script accepts as commandline parameter the name of the project
to be opened.
'The absolute path need to b included. Ex:
'wscript select~1.vbs "C:\testprojects\test1.mcp"
'If no command line arguments is given, the script prompts for the user
for the
'absolute path of the project file to be opened.
'If specified the script tries to open the project, else opens te
default one "c:\testprojects\test1.mcp'
'
'This script opens the project and selects the files (if on filelist
pane or link order pane) that belong to the
'default target
```

```
option explicit

'*******Variable declaration
dim codewarrior
dim project

dim projectname
dim targetIntf
dim count
dim projectCollection
dim targetcollection
dim result
dim showinputbox
dim objArgs
dim filespec
dim filename

'****** Script ********
Set objArgs = Wscript.Arguments

if objArgs.Count > 1 then
MsgBox "This Script expects only one argument, rest of the arguments
will be ignored!!"
showinputbox = false
projectname = CStr(objArgs(0))
end if

if objArgs.Count = 0 then
showinputbox = true
else
showinputbox = false
projectname = CStr(objArgs(0))
end if

if showinputbox = true then
result = InputBox("Enter the absolute path for the project to be
opened","Input", projectname, 100, 100)

If result = "" Then
projectname = "c:\testprojects\test1.mcp"
else
projectname = cstr(result)
end if
end if

'Create automation app object
set codewarrior = CreateObject("CodeWarrior.CodeWarriorApp")
```

```
MsgBox "App Created"

project = Null
'open project
set project = codewarrior.OpenProject(projectname, true, 2, 0 )

'Execute Menu Command cmd_Make
'To change the commad refer the CodeWarriorCommandNumbers.h file
codewarrior.DoCommand  1406
```

## Getting the Active Document

The `CodeWarriorApp` offers the `get_ActiveDocument` method to obtain the currently active document in the CodeWarrior application.

### Syntax

```
virtual HRESULT get_ActiveDocument(

ICodeWarriorDocument **pval) = 0;
```

where,

- `pval` contains the address of a pointer to the active document in the CodeWarrior application.

# Index