

# **CodeWarrior™**

# **Development Tools**

## **Extending the**

## **CodeWarrior IDE**

Revised: 4/19/02

Metrowerks, the Metrowerks insignia, and CodeWarrior are registered trademarks of Metrowerks Corp. in the US and/or other countries. All other trade names, trademarks and registered trademarks are the property of their respective owners.

© Copyright. 2002. Metrowerks Corp. ALL RIGHTS RESERVED.

Metrowerks reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Metrowerks does not assume any liability arising out of the application or use of any product described herein.

Metrowerks 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 or automobile navigation or communication, emergency systems, or other applications with a similar degree of potential hazard.

Documentation stored on electronic media may be printed for personal use only. Except for the forgoing, no portion of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Metrowerks.

ALL SOFTWARE, DOCUMENTATION AND RELATED MATERIALS ARE SUBJECT TO THE METROWERKS END USER LICENSE AGREEMENT FOR SUCH PRODUCT.

## **How to Contact Metrowerks:**

---

**Corporate Headquarters** Metrowerks Corporation  
9801 Metric Blvd.  
Austin, TX 78758  
U.S.A.

**World Wide Web** <http://www.metrowerks.com>

**Ordering & Technical Support** Voice: (800) 377-5416  
Fax: (512) 997-4901

---

# Table of Contents

---

<b>1 Overview</b>	<b>25</b>
Introduction to Extending the CodeWarrior IDE . . . . .	26
Read the Release Notes . . . . .	30
New Things . . . . .	31
Changes for Version 13 . . . . .	32
Changes for Version 12 . . . . .	32
Changes in Version 11. . . . .	34
Changes in Version 8 . . . . .	35
Requirements for Developing Plug-ins . . . . .	35
Installing CodeWarrior Software . . . . .	36
Installing the Plug-in API SDK . . . . .	36
What You Should Already Know . . . . .	36
Starting Points . . . . .	37
Getting started. . . . .	37
Other Resources . . . . .	38
<b>2 Plug-in Development Setup</b>	<b>39</b>
Overview . . . . .	39
About the SDK . . . . .	39
SDK Overview. . . . .	40
SDK Contents . . . . .	40
Resources for Scripting . . . . .	41
Getting Started . . . . .	41
Installation Steps. . . . .	41
Plug-in Installation . . . . .	44
Plug-ins Folder . . . . .	44
Automatic Installation . . . . .	45
Plug-in Debugging . . . . .	46
Debugging Setup. . . . .	46
Plug-in Debugging Options . . . . .	48
Troubleshooting . . . . .	51
<b>3 IDE and Plug-in Architecture</b>	<b>53</b>
Overview . . . . .	53
Types of Plug-ins . . . . .	54

---

## Table of Contents

---

Compiler, Pre-linker, Linker, and Post-linker Plug-ins . . . . .	54
Settings Panel Plug-ins . . . . .	55
Version Control Plug-ins . . . . .	55
Third Party Editors (Mac OS) . . . . .	56
COM Servers (Mac and UNIX OS) . . . . .	56
<b>4 Creating CodeWarrior IDE Plug-ins</b>	<b>57</b>
Overview . . . . .	57
Plug-in Binary Formats and Installation . . . . .	57
Plug-in Binary Formats . . . . .	58
Platform Differences in Plug-in API . . . . .	61
Resources on Mac OS . . . . .	61
Platform Dependent API Differences . . . . .	62
Getting Runtime Information . . . . .	64
Informational Entry Points . . . . .	65
Specifying Plug-in Capabilities . . . . .	65
Specifying a Plug-in's Dropin and Display Names . . . . .	70
Specifying Associated Settings Panels . . . . .	72
Specifying Panel Family . . . . .	73
Responding to IDE Requests . . . . .	75
Basic Request Handling . . . . .	75
Returning Error Results to the IDE . . . . .	77
Common Requests . . . . .	78
Reentrancy Considerations . . . . .	80
Managing Memory . . . . .	80
Kinds of Memory . . . . .	81
Pointers . . . . .	81
Allocating and Disposing Pointers . . . . .	81
Handles . . . . .	82
Accessing Handle Memory . . . . .	83
Allocating and Disposing Handles . . . . .	84
Resizing Handles . . . . .	85
Pointers versus Handles . . . . .	85
Managing Files . . . . .	85
Project Structure . . . . .	86
Specifying Project Components . . . . .	87
File Specifications . . . . .	87

Obtaining Information About Files . . . . .	87
Getting a File . . . . .	88
Getting Files Not in the Project . . . . .	88
Modifying Files in the Project . . . . .	88
Getting Information About Projects . . . . .	89
Getting Information About Targets . . . . .	89
Getting Information About Overlays and Segments . . . . .	89
Getting Information About Project Paths . . . . .	90
Managing Plug-in Data . . . . .	91
Storing and Retrieving Plug-in Data . . . . .	91
Storing and Retrieving Preference Data . . . . .	92
Storing Data Externally . . . . .	94
Handling User Interaction . . . . .	94
Showing an Editor Window . . . . .	95
Reporting Progress . . . . .	95
Reporting Errors and Warnings from Compilers and Linkers .	95
Reporting Other User Messages . . . . .	96
Displaying Plug-in Dialogs . . . . .	96
Checking for User Breaks . . . . .	96
Obsolete User Interface Routines . . . . .	97
Error Handling . . . . .	97
General Error Handling . . . . .	97
Obtaining Error Information . . . . .	97
Reporting Errors to the IDE . . . . .	98
<b>5 Plug-in API Reference</b>	<b>99</b>
Overview . . . . .	99
Routines for Plug-ins . . . . .	99
Plug-in Context . . . . .	99
Alphabetical Routine Index . . . . .	99
Functional Routine Index . . . . .	101
CWAddProjectEntry . . . . .	104
CWAlert . . . . .	105
CWAllocateMemory . . . . .	106
CWAllocMemHandle . . . . .	107
CWCreateNewTextDocument . . . . .	108
CWDonePluginRequest . . . . .	109

---

## Table of Contents

---

CWFindAndLoadFile . . . . .	110
CWFreeMemHandle . . . . .	112
CWFreeMemory . . . . .	113
CWGetAccessPathInfo . . . . .	113
CWGetAccessPathListInfo. . . . .	114
CWGetAccessPathSubdirectory . . . . .	115
CWGetAPIVersion . . . . .	116
CWGetCallbackOSError. . . . .	117
CWGetCOMApplicationInterface . . . . .	118
CWGetCOMProjectInterface . . . . .	118
CWGetCOMDesignInterface. . . . .	119
CWGetCOMTargetInterface . . . . .	119
CWGetFileInfo. . . . .	120
CWGetFileText. . . . .	121
CWGetFrameworkCount . . . . .	122
CWGetFrameworkInfo . . . . .	123
CWGetFrameworkSharedLibrary . . . . .	123
CWGetIDEInfo . . . . .	124
CWGetMemHandleSize. . . . .	124
CWGetNamedPreferences. . . . .	125
CWGetOutputFileDialog . . . . .	126
CWGetOverlay1FileInfo. . . . .	126
CWGetOverlay1GroupInfo . . . . .	127
CWGetOverlay1GroupsCount . . . . .	128
CWGetOverlay1Info . . . . .	129
CWGetPluginData . . . . .	129
CWGetPluginRequest. . . . .	130
CWGetProjectFile . . . . .	131
CWGetProjectFileCount. . . . .	132
CWGetSegmentInfo . . . . .	132
CWGetTargetDataDirectory . . . . .	133
CWGetTargetName. . . . .	134
CWLockMemHandle . . . . .	134
CWMacOSErrToCWResult . . . . .	136
CWPostDialog . . . . .	136
CWPostFileAction . . . . .	137
CWPreDialog . . . . .	138

CWPreFileAction . . . . .	138
CWReleaseFileText . . . . .	139
CWRemoveProjectEntry . . . . .	139
CWReportMessage . . . . .	140
CWResizeMemHandle . . . . .	141
CWResolveRelativePath. . . . .	142
CWSetModDate . . . . .	143
CWSetPluginOSError. . . . .	144
CWShowStatus . . . . .	145
CWStorePluginData . . . . .	145
CWUnlockMemHandle . . . . .	147
CWUserBreak . . . . .	148
User Routines for Plug-ins . . . . .	148
Main Plug-in Entry Point . . . . .	149
CWPlugin_GetDropInFlags Entry Point . . . . .	150
CWPlugin_GetDropInName Entry Point . . . . .	151
CWPlugin_GetDisplayName Entry Point . . . . .	151
CWPlugin_GetPluginInfo . . . . .	152
Data Structures for Plug-ins . . . . .	153
CWAcessPathInfo . . . . .	154
CWAcessPathListInfo . . . . .	155
CWAcessPathType. . . . .	156
CWAddr64 . . . . .	157
CWDataType . . . . .	157
CWDependencyType . . . . .	158
CWFileInfo . . . . .	159
CWFileName . . . . .	162
CWFileSpec . . . . .	162
CWFileType . . . . .	164
CWFrameworkInfo. . . . .	165
CWIDEInfo . . . . .	165
CWMemHandle . . . . .	166
CWMessagRef . . . . .	167
CWNewProjectEntryInfo . . . . .	168
CWNewTextDocumentInfo . . . . .	170
CWOSResult . . . . .	170
CWOvrlay1FileInfo . . . . .	171

## Table of Contents

---

CWOverlay1GroupInfo . . . . .	171
CWOverlay1Info . . . . .	172
CWPluginContext . . . . .	172
CWPluginInfo . . . . .	173
CWProjectFileInfo . . . . .	174
CWProjectSegmentInfo . . . . .	178
CWRelativePath . . . . .	178
CWRelativePathFormat . . . . .	180
CWRelativePathTypes . . . . .	181
CWResult . . . . .	182
DropInFlags . . . . .	182
Constants for Plug-ins . . . . .	184
CWDROPINCOMPILERTYPE . . . . .	185
CWDROPINLINKERTYPE . . . . .	185
CWDROPINPREFSTYPE . . . . .	186
CWDROPINPREFSTYPE_1 . . . . .	186
CWDROPINVCCSTYPE . . . . .	187
cwFileTypePrecompiledHeader . . . . .	187
cwFileTypeText . . . . .	187
cwFileTypeUnknown . . . . .	188
cwSystemPath . . . . .	188
cwUserPath . . . . .	188
kCurrentCompiledFile . . . . .	189
kDefaultLinkPosition . . . . .	189
kTargetGlobalPluginData . . . . .	189
messagetypeInfo . . . . .	190
messagetypeWarning . . . . .	190
messagetypeError . . . . .	190
reqAbout . . . . .	191
reqIdle . . . . .	191
reqInitialize . . . . .	192
reqPrefsChange . . . . .	192
reqTerminate . . . . .	193
Result Codes for Plug-ins . . . . .	193
cwErrCantSetAttribute . . . . .	194
cwErrFileNotFoundException . . . . .	194
cwErrInvalidCallback . . . . .	194

---

cwErrInvalidMPCallback . . . . .	195
cwErrInvalidParameter . . . . .	195
cwErrOSError . . . . .	195
cwErrOutOfMemory . . . . .	196
cwErrRequestFailed . . . . .	196
cwErrSilent . . . . .	196
cwErrStringBufferOverflow . . . . .	197
cwErrUnknownFile. . . . .	197
cwErrUserCanceled . . . . .	197
cwNoErr . . . . .	197
<b>6 Creating Compiler and Linker Plug-ins</b>	<b>199</b>
Overview . . . . .	199
Compiler and Linker DropIn Flags . . . . .	199
Compiler Capability Flags. . . . .	200
Linker Capability Flags . . . . .	203
Compiler and Linker-Specific Entry Points . . . . .	206
Specifying File Mappings . . . . .	206
Specifying Target Platforms . . . . .	208
Compiler Browser Symbol Entry Point . . . . .	209
Linker Symbol Unmangling Entry Point. . . . .	210
Handling Compiler and Linker Requests . . . . .	211
Compiling. . . . .	211
Linking . . . . .	213
Providing Target Information . . . . .	216
Disassembling . . . . .	219
Generating Browser Data . . . . .	222
Generating Debugging Data . . . . .	222
Additional Compiler and Linker Requests. . . . .	223
Managing Projects and Targets . . . . .	226
Adding a File to a Project . . . . .	226
Removing a File from a Project. . . . .	227
Storing Object, Resource, and Class Browser Data . . . . .	227
Getting Object Data and Resource Data . . . . .	228
Specifying File Dependencies . . . . .	229
Caching File Data . . . . .	230
Managing Target Storage . . . . .	230

---

## Table of Contents

---

Storing Linker Output . . . . .	234
Storing Post-Linker Output . . . . .	235
Getting Linker Output . . . . .	235
Getting Runtime Settings . . . . .	235
<b>7 Compiler and Linker Plug-in Reference</b>	<b>237</b>
Overview . . . . .	237
Routines for Compiler and Linker Plug-ins . . . . .	237
Alphabetical Routine Index . . . . .	237
Functional Routine Index . . . . .	238
CWCachePrecompiledHeader . . . . .	240
CWDisplayLines . . . . .	241
CWFreeObjectData . . . . .	242
CWGetBrowseOptions . . . . .	242
CWGetBuildSequenceNumber . . . . .	243
CWGetCommandLineArgs . . . . .	244
CWGetEnvironmentVariable . . . . .	244
CWGetEnvironmentVariableCount . . . . .	246
CWGetMainFileID . . . . .	247
CWGetMainFileName . . . . .	248
CWGetMainFileSpec . . . . .	249
CWGetMainFileText . . . . .	249
CWGetModifiedFiles . . . . .	250
CWGetPrecompiledHeaderSpec . . . . .	251
CWGetResourceFile . . . . .	252
CWGetStoredObjectFileSpec . . . . .	253
CWGetSuggestedObjectFileSpec . . . . .	254
CWGetTargetInfo . . . . .	255
CWGetTargetStorage . . . . .	256
CWGetWorkingDirectory . . . . .	257
CWIsAutoPrecompiling . . . . .	258
CWIsCachingPrecompiledHeaders . . . . .	260
CWIsGeneratingDebugInfo . . . . .	260
CWIsPrecompiling . . . . .	261
CWIsPreprocessing . . . . .	262
CWLoadObjectData . . . . .	263
CWPutResourceFile . . . . .	264

CWSetTargetInfo . . . . .	265
CWSetTargetStorage . . . . .	266
CWStoreObjectData . . . . .	267
User Routines for Compiler and Linker Plug-ins . . . . .	268
CWPlugin_GetDefaultMappingList Entry Point . . . . .	269
CWPlugin_GetTargetList Entry Point . . . . .	270
Helper_GetCompilerBrSymbols Entry Point . . . . .	270
Helper_Unmangle Entry Point . . . . .	272
Data Structures for Compiler and Linker Plug-ins . . . . .	274
CWBrowseOptions . . . . .	274
CWCompilerBrSymbol . . . . .	276
CWCompilerBrSymbolInfo . . . . .	276
CWCompilerBrSymbolList . . . . .	277
CWDependencyInfo . . . . .	278
CWExtensionMapping . . . . .	279
CWExtMapList . . . . .	280
CWOBJECTDATA . . . . .	281
CWTargetInfo . . . . .	283
CWTARGETLIST . . . . .	289
CWUnmangleInfo . . . . .	290
Constants for Compiler and Linker Plug-ins . . . . .	292
cantDisassemble . . . . .	295
cwAccessAbsolute . . . . .	296
cwAccessPathRelative . . . . .	296
cwAccessFileName . . . . .	296
cwAccessFileRelative . . . . .	297
DROPINCOMPILERLINKERAPIVERSION . . . . .	297
exelinkageFlat . . . . .	297
exelinkageOverlay1 . . . . .	298
exelinkageSegmented . . . . .	298
isPostLinker . . . . .	299
isPreLinker . . . . .	299
kCandidassemble . . . . .	299
kCanimport . . . . .	299
kCanprecompile . . . . .	300
kCanpreprocess . . . . .	300
kCompAllowDupFileNames . . . . .	300

---

## Table of Contents

---

kCompAlwaysReload . . . . .	301
kCompEmitsOwnBrSymbols . . . . .	301
kCompMultiTargAware . . . . .	302
kCompReentrant . . . . .	302
kCompRequiresFileListBuildStartedMsg . . . . .	302
kCompRequiresProjectBuildStartedMsg . . . . .	302
kCompRequiresSubProjectBuildStartedMsg . . . . .	303
kCompRequiresTargetBuildStartedMsg . . . . .	303
kCompRequiresTargetCompileStartedMsg . . . . .	303
kCompSavesDbgPreprocess . . . . .	303
kCompUsesTargetStorage . . . . .	304
kGeneratescode . . . . .	304
kGeneratesrsrcs . . . . .	304
kIgnored . . . . .	304
kLaunchable . . . . .	305
kIsMPAware . . . . .	305
kIspascal . . . . .	305
kPersistent . . . . .	305
kPrecompile . . . . .	306
kRsrcfile . . . . .	306
linkAllowDupFileNames . . . . .	306
linkAlwaysReload . . . . .	307
linkMultiTargAware . . . . .	307
linkOutputNone . . . . .	307
linkOutputFile . . . . .	308
linkOutputDirectory . . . . .	308
linkRequiresFileListBuildStartedMsg . . . . .	308
linkRequiresProjectBuildStartedMsg . . . . .	309
linkRequiresSubProjectBuildStartedMsg . . . . .	309
linkRequiresTargetBuildStartedMsg . . . . .	309
linkRequiresTargetLinkStartedMsg . . . . .	310
linkerDisasmRequiresPreprocess . . . . .	310
linkerGetTargetInfoThreadSafe . . . . .	310
linkerInitializeOnMainThread . . . . .	310
linkerSuggestsNonRecursiveAccessPaths . . . . .	311
linkerUnmangles . . . . .	311
linkerUsesCaseInsensitiveSymbols . . . . .	311

---

linkerUsesFrameworks . . . . .	311
linkerUsesTargetStorage. . . . .	312
linkerWantsPreRunRequest . . . . .	312
magicCapLinker . . . . .	312
reqCheckSyntax . . . . .	312
reqCompile . . . . .	313
reqCompDisassemble. . . . .	313
reqDisassemble . . . . .	314
reqFileListBuildEnded . . . . .	315
reqFileListBuildStarted . . . . .	315
reqLink . . . . .	315
reqMakeParse . . . . .	316
reqPreprocessForDebugger . . . . .	316
reqPreRun. . . . .	316
reqProjectBuildEnded. . . . .	316
reqProjectBuildStarted . . . . .	317
reqSubProjectBuildEnded . . . . .	317
reqSubProjectBuildStarted. . . . .	318
reqTargetBuildEnded . . . . .	318
reqTargetBuildStarted. . . . .	318
reqTargetCompileEnded . . . . .	319
reqTargetCompileStarted . . . . .	319
reqTargetInfo . . . . .	319
reqTargetLinkEnded . . . . .	320
reqTargetLinkStarted . . . . .	320
reqTargetLoaded. . . . .	320
reqTargetPrefsChanged . . . . .	321
reqTargetUnloaded. . . . .	321
targetCPU68K . . . . .	322
targetCPUAny. . . . .	323
targetCPUEmbeddedPowerPC. . . . .	323
targetCPUi80x86 . . . . .	323
targetCPUMips . . . . .	324
targetCPUNECv800 . . . . .	324
targetCPUPowerPC . . . . .	325
targetOSAny . . . . .	325
targetOSEmbeddedABI . . . . .	326

---

---

## Table of Contents

---

targetOSMacintosh . . . . .	326
targetOSMagicCap . . . . .	326
targetOSOS9. . . . .	327
targetOSWindows . . . . .	327
Result Codes for Compiler and Linker Plug-ins . . . . .	328
cwErrObjectFileNotStored. . . . .	328
cwErrUnknownSegment . . . . .	328
<b>8 Creating Settings Panel Plug-ins</b>	<b>331</b>
Creating Settings Panel Plug-ins Overview . . . . .	331
Platform Differences in Settings Panel API . . . . .	332
Settings Panel Context . . . . .	333
Differences in Common Panel Routines . . . . .	337
Basic Settings Panel Operation . . . . .	338
Settings Panel Overview . . . . .	338
Settings Panel Entry Points . . . . .	338
PanelFlags Structure . . . . .	340
Settings Panel Flags . . . . .	341
Settings Panel Data . . . . .	342
Settings Panel Requests . . . . .	345
Handling Settings Panel Requests . . . . .	345
Determining the “Hit” Dialog Item . . . . .	350
Returning Errors to the IDE . . . . .	351
Initialization and Shutdown . . . . .	353
Getting and Putting Settings Data . . . . .	355
Manipulating Settings Data . . . . .	358
Handling User Interaction. . . . .	365
Importing and Exporting Settings Data . . . . .	366
Managing Panel Items. . . . .	371
Getting and Setting Control Values . . . . .	371
Enabling and Disabling Items . . . . .	372
Validating Input . . . . .	373
Managing Input Focus . . . . .	373
Redrawing Panel Items . . . . .	374
<b>9 Creating Settings Panel Plug-ins on Windows</b>	<b>375</b>
Creating Settings Panel Plug-ins on Windows Overview . . . . .	375

---

Help Information Entry Point . . . . .	375
Handling Panel Requests from the IDE . . . . .	376
Handling Panel Requests . . . . .	376
Creating a Panel's Interface . . . . .	378
Panel Resource Construction . . . . .	378
Dialog Resource Construction . . . . .	378
Control ID Assignment . . . . .	380
Panel Layout Requirements . . . . .	381
Supported Control Types . . . . .	381
Supported Control Styles . . . . .	382
<b>10 Creating Settings Panel Plug-ins on Mac OS</b>	<b>383</b>
Creating Settings Panel Plug-ins on Mac OS Overview. . . . .	383
Settings Panel Resources. . . . .	384
User Interface Resources . . . . .	384
Providing Balloon Help . . . . .	387
Apple Event Dictionary ('aete') Resource . . . . .	387
Requests from the IDE. . . . .	388
Getting a Panel Request . . . . .	389
Returning Panel Results to the IDE . . . . .	389
Handling Keyboard and Mouse Hits . . . . .	389
Filtering Low-Level Events . . . . .	391
Responding to Item Hits . . . . .	391
Drawing Custom Items . . . . .	392
Managing Input Focus (Mac OS) . . . . .	393
Handling Edit Menu Commands . . . . .	393
Handling Drag And Drop . . . . .	395
Handling Apple Events . . . . .	396
<b>11 Settings Panel Plug-in API Reference</b>	<b>399</b>
Settings Panel Plug-in API Reference Overview. . . . .	399
Routines for Settings Panel Plug-ins. . . . .	399
Organization of Function Reference. . . . .	400
Plug-in Context . . . . .	400
Alphabetical Routine Index . . . . .	401
Functional Routine Index . . . . .	404
CWGetArraySettingElement. . . . .	408

---

## Table of Contents

---

CWGetArraySettingSize . . . . .	410
CWGetBooleanValue . . . . .	411
CWGetFloatingPointValue. . . . .	411
CWGetIntegerValue . . . . .	412
CWGetNamedSetting. . . . .	413
CWGetRelativePathValue . . . . .	414
CWGetStringValue . . . . .	415
CWGetStructureSettingField. . . . .	416
CWPanelActivateItem . . . . .	418
CWPanelAppendItems . . . . .	419
CWPanelChooseRelativePath . . . . .	420
CWPanelDeleteListItem. . . . .	422
CWPanelEnableItem . . . . .	423
CWPanelGetCurrentPrefs . . . . .	423
CWPanelGetDebugFlag. . . . .	424
CWPanelGetDialogItemHit . . . . .	425
CWPanelGetFactoryPrefs . . . . .	426
CWPanelGetData . . . . .	427
CWPanelGetItemMaxLength . . . . .	428
CWPanelGetItemText . . . . .	429
CWPanelGetItemTextHandle . . . . .	430
CWPanelGetItemValue . . . . .	431
CWPanelGetListItemText . . . . .	432
CWPanelGetNumBaseDialogItems . . . . .	433
CWPanelGetOriginalPrefs. . . . .	434
CWPanelGetPanelPrefs . . . . .	435
CWPanelGetRelativePathString . . . . .	436
CWPanelInsertListItem . . . . .	438
CWPanelInvalItem . . . . .	439
CWPanelSetFactoryFlag. . . . .	440
CWPanelSetItemData . . . . .	441
CWPanelSetItemMaxLength. . . . .	441
CWPanelSetItemText . . . . .	442
CWPanelSetTextHandle. . . . .	443
CWPanelSetItemValue . . . . .	444
CWPanelSetListItemText . . . . .	445
CWPanelSetRecompileFlag . . . . .	446

---

## Table of Contents

---

CWPanelSetRelinkFlag . . . . .	447
CWPanelSetReparseFlag . . . . .	448
CWPanelSetResetPathsFlag . . . . .	449
CWPanelSetRevertFlag . . . . .	449
CWPanelShowItem. . . . .	450
CWPanelValidItem . . . . .	451
CWPanlActivateItem . . . . .	452
CWPanlAppendItems. . . . .	452
CWPanlChooseRelativePath. . . . .	452
CWPanlDrawPanelBox . . . . .	452
CWPanlDrawUserItemBox . . . . .	453
CWPanlEnableItem. . . . .	453
CWPanlGetArraySettingElement. . . . .	454
CWPanlGetArraySettingSize. . . . .	454
CWPanlGetBooleanValue . . . . .	454
CWPanlGetFloatingPointValue. . . . .	454
CWPanlGetIntegerValue . . . . .	455
CWPanlGetItemControl. . . . .	455
CWPanlGetData . . . . .	456
CWPanlGetItemMaxLength . . . . .	456
CWPanlGetItemRect . . . . .	456
CWPanlGetItemText . . . . .	457
CWPanlGetItemTextHandle . . . . .	457
CWPanlGetValue. . . . .	457
CWPanlGetMacPort . . . . .	458
CWPanlGetNamedSetting. . . . .	458
CWPanlGetPanelPrefs . . . . .	458
CWPanlGetRelativePathString . . . . .	459
CWPanlGetRelativePathValue . . . . .	459
CWPanlGetStringValue . . . . .	459
CWPanlGetStructureSettingField. . . . .	459
CWPanlInstallUserItem . . . . .	460
CWPanlInvalItem . . . . .	460
CWPanlReadBooleanSetting . . . . .	460
CWPanlReadFloatingPointSetting . . . . .	460
CWPanlReadIntegerSetting . . . . .	461
CWPanlReadRelativePathAEDesc . . . . .	461

---

## Table of Contents

---

CWPanlReadRelativePathSetting . . . . .	462
CWPanlReadStringSetting . . . . .	462
CWPanlRemoveUserItem . . . . .	462
CWPanlSetBooleanValue . . . . .	462
CWPanlSetFloatingPointValue . . . . .	463
CWPanlSetIntegerValue . . . . .	463
CWPanlSetItemData . . . . .	463
CWPanlSetItemMaxLength . . . . .	463
CWPanlSetItemText . . . . .	464
CWPanlSetItemTextHandle . . . . .	464
CWPanlSetItemValue . . . . .	464
CWPanlSetRelativePathValue . . . . .	464
CWPanlSetStringValue . . . . .	464
CWPanlShowItem . . . . .	465
CWPanlValidItem . . . . .	465
CWPanlWriteBooleanSetting . . . . .	465
CWPanlWriteFloatingPointSetting . . . . .	465
CWPanlWriteIntegerSetting . . . . .	466
CWPanlWriteRelativePathAEDesc . . . . .	466
CWPanlWriteRelativePathSetting . . . . .	467
CWPanlWriteStringSetting . . . . .	467
CWReadBooleanSetting . . . . .	467
CWReadFloatingPointSetting . . . . .	468
CWReadIntegerSetting . . . . .	469
CWReadRelativePathSetting . . . . .	470
CWReadStringSetting . . . . .	470
CWSetBooleanValue . . . . .	471
CWSetFloatingPointValue . . . . .	472
CWSetIntegerValue . . . . .	474
CWSetRelativePathValue . . . . .	475
CWSetValue . . . . .	476
CWWriteBooleanSetting . . . . .	477
CWWriteFloatingPointSetting . . . . .	477
CWWriteIntegerSetting . . . . .	478
CWWriteRelativePathSetting . . . . .	479
CWWriteStringSetting . . . . .	479
User Routines for Settings Panel Plug-ins . . . . .	480

---

CWPlugin_GetHelpInfo Entry Point . . . . .	480
Data Structures for Settings Panel Plug-ins . . . . .	481
CWDIALOG . . . . .	481
CWHelpInfo . . . . .	482
CWSettingID . . . . .	482
PanelFlags . . . . .	483
PanelParamBlkPtr . . . . .	485
PanelParameterBlock . . . . .	485
Constants for Settings Panel Plug-ins . . . . .	492
CW_STRICT_DIALOGS . . . . .	494
DROPINPANELAPIVERSION . . . . .	494
kCurrentCWHelpInfoVersion . . . . .	495
menu_Clear . . . . .	495
menu_Copy . . . . .	495
menu_Cut . . . . .	496
menu_Paste . . . . .	496
menu_SelectAll . . . . .	496
panelScopeGlobal . . . . .	497
panelScopeProject . . . . .	497
panelScopeTarget . . . . .	498
reqActivateItem . . . . .	498
reqAEGetPref . . . . .	499
reqAESetPref . . . . .	499
reqByteSwapData . . . . .	500
reqDeactivateItem . . . . .	500
reqDragDrop . . . . .	501
reqDragEnter . . . . .	502
reqDragExit . . . . .	503
reqDragWithin . . . . .	504
reqDrawCustomItem . . . . .	504
reqFilter . . . . .	505
reqFindStatus . . . . .	505
reqFirstLoad . . . . .	506
reqGetData . . . . .	506
reqGetFactory . . . . .	507
reqHandleClick . . . . .	508
reqHandleKey . . . . .	508

---

## Table of Contents

---

reqInitDialog . . . . .	509
reqInitPanel . . . . .	509
reqItemHit . . . . .	510
reqObeyCommand . . . . .	511
reqPutData . . . . .	512
reqReadSettings . . . . .	512
reqRenameProject . . . . .	513
reqSetupDebug . . . . .	513
reqTermDialog . . . . .	514
reqTermPanel . . . . .	514
reqUpdatePref . . . . .	515
reqValidate . . . . .	516
reqWriteSettings . . . . .	517
supportsByteSwapping . . . . .	517
supportsTextSettings . . . . .	518
usesStrictAPI . . . . .	518
Settings Panel Result Codes . . . . .	519
kBadPrefVersion . . . . .	519
kMissingPrefErr . . . . .	519
kSettingNotFoundErr . . . . .	519
kSettingTypeMismatchErr . . . . .	520
kInvalidCallbackErr . . . . .	520
kSettingOutOfRangeErr . . . . .	520
<b>12 Version Control System Plug-in API . . . . .</b>	<b>523</b>
What's New in the VCS API? . . . . .	523
Overview . . . . .	524
Operational Phases . . . . .	525
VCS Plug-in Interface . . . . .	527
The Main Entry Point . . . . .	527
Plug-in Callbacks . . . . .	538
Standard Callbacks . . . . .	539
VCS Commands . . . . .	554
VCS Requests . . . . .	556
Initialization / Termination . . . . .	557
Database Connection / Disconnection . . . . .	558
Queries . . . . .	559

---

Information . . . . .	559
File Processing . . . . .	560
VCS API Version 1 Compatibility . . . . .	565
Version 1 VCS API . . . . .	566
Porting a Version 1 VCS Plug-in to Version 7 or Later API . .	569
<b>13 Browser Reference</b>	<b>573</b>
Browser Reference Overview. . . . .	573
Browser Records . . . . .	573
Browse Header . . . . .	574
Browser Data Stream . . . . .	575
Fields For All Records . . . . .	576
Additional Fields For Functions . . . . .	579
Additional Fields For Classes . . . . .	580
Additional Field For Templates . . . . .	584
Additional Field For Global Variables . . . . .	584
Data Structures for the Browser. . . . .	585
BrowseHeader . . . . .	585
Constants for the Browser . . . . .	587
BROWSE_EARLIEST_COMPATIBLE_VERSION . . . . .	587
BROWSE_HEADER . . . . .	588
BROWSE_VERSION . . . . .	588
EAccess . . . . .	588
EBrowserItem . . . . .	589
ELanguage . . . . .	590
EMember . . . . .	591
ETemplateType . . . . .	592
<b>14 PowerPC Object Code (Mac OS)</b>	<b>593</b>
PowerPC Object Code Overview . . . . .	593
PowerPC Object Code Structure . . . . .	594
PowerPC Object Header . . . . .	594
PowerPC Object Data Section . . . . .	597
Preventing Dead-Stripping . . . . .	598
PowerPC Simple Hunks. . . . .	598
PowerPC Regular Code Hunks. . . . .	599
PowerPC Data Hunks. . . . .	601

---

## Table of Contents

---

PowerPC Alternate Entry Point Hunks . . . . .	603
PowerPC Cross-Reference Hunks . . . . .	604
PowerPC PEF Import Hunks. . . . .	605
PowerPC Source File Specification Hunks . . . . .	607
PowerPC Object Pascal Hunks . . . . .	608
PowerPC Reserved Hunks. . . . .	610
PowerPC Symbolic Data Header . . . . .	610
PowerPC Symbolic Function Data Section . . . . .	611
PowerPC Symbolic Type Data Section . . . . .	614
Other Types for PowerPC . . . . .	615
PowerPC Pointer Type . . . . .	617
PowerPC Array Type . . . . .	618
PowerPC Structured Type . . . . .	618
PowerPC Enumerated Type . . . . .	619
PowerPC Pascal Array Type . . . . .	621
PowerPC Pascal Subrange Type . . . . .	622
PowerPC Pascal Set Type . . . . .	623
PowerPC Pascal Enumerated Type . . . . .	623
PowerPC Pascal String Type . . . . .	624
PowerPC Name Table Section . . . . .	625
<b>15 Library Specification (Mac OS)</b>	<b>627</b>
Library Specification Overview. . . . .	627
Overall Structure . . . . .	628
Library Header. . . . .	628
File Data Records . . . . .	629
File and Path Names . . . . .	630
Object Code Modules . . . . .	630
<b>16 External Editors on Mac OS</b>	<b>631</b>
External Editors on Mac OS Overview. . . . .	631
Open Document . . . . .	632
Get Text. . . . .	634
Window Search . . . . .	635
Modified (from IDE to Editor) . . . . .	636
Modified (from Editor to IDE) . . . . .	636
Sending Other Events to the IDE . . . . .	638

---

Structure Alignment for External Editors . . . . .	638
Using an External Editor . . . . .	639
<b>17 Mac OS Plug-in Resource Formats</b>	<b>641</b>
Resources for Compiler and Linker Plug-ins (Mac OS) . . . . .	641
Resource Descriptions . . . . .	642
'cfrg' Resource and PowerPC Executable Code . . . . .	642
'Dhlp' Resource . . . . .	642
'Drop' Resource . . . . .	643
'EMap' Resource . . . . .	644
'Flag' Resource . . . . .	645
'Fmly' Resource . . . . .	650
'Targ' Resource . . . . .	650
'STR' Resource . . . . .	652
'STR#' Resource . . . . .	652
<b>18 Mac OS CodeWarrior Scripting</b>	<b>655</b>
CodeWarrior Mac OS Scripting Overview . . . . .	655
AppleScript Tools and Reference Material . . . . .	656
Writing Your First CodeWarrior IDE AppleScript . . . . .	657
CodeWarrior IDE AppleScript Events . . . . .	658
Processing Errors . . . . .	659
Required Events . . . . .	661
File Handling Events . . . . .	662
Building Events . . . . .	666
Status/Query Events . . . . .	674
Navigation Events . . . . .	682
CodeWarrior IDE AppleScript Classes . . . . .	683
Project Classes . . . . .	684
Compiler Classes . . . . .	693
CodeGen Classes . . . . .	699
Disassembler Classes . . . . .	703
Linker Classes . . . . .	704
Build Classes . . . . .	710
Browser Classes . . . . .	711
Editor Classes . . . . .	712
Object Classes . . . . .	719

---

## Table of Contents

---

Miscellaneous Classes . . . . .	722
Coding with CodeWarrior IDE and Apple Events . . . . .	729
<b>19 CodeWarrior Scripting on Microsoft Windows</b>	<b>731</b>
CodeWarrior Windows Scripting Overview . . . . .	731
Tools and Reference Material . . . . .	732
Microsoft Scripting Technologies Web Site . . . . .	732
OLE/COM Object Viewer . . . . .	733
Magazine Articles . . . . .	735
Published Books . . . . .	736
Sample Scripts and How to Get Started . . . . .	736
How to Start Scripting . . . . .	736
A Word About Collections . . . . .	739
Script Examples . . . . .	740
<b>20 Perl Scripting</b>	<b>745</b>
Installing the Perl Software Plug-ins . . . . .	745
Windows Perl Installation . . . . .	746
Mac OS Perl Installation . . . . .	746
Configuring the Perl Target Settings Panel . . . . .	747
Perl Scripting . . . . .	749
Special Considerations . . . . .	750
StdIn Usage . . . . .	750
Avoiding Link Errors . . . . .	750
Opening Input and Output Files . . . . .	751
Manipulating the IDE via Perl . . . . .	751
Forcing a Perl Script to Run on Every Build . . . . .	751
MacPerl and AppleScript . . . . .	752
<b>21 Frequently Asked Questions (FAQ)</b>	<b>755</b>
Questions and Answers . . . . .	755
<b>Index</b>	<b>757</b>

# Overview

---

Welcome to *Extending the CodeWarrior IDE*. This manual provides explanatory material on extending the standard features of the CodeWarrior IDE (Integrated Development Environment), through scriptable automation and enhancement plug-ins.

It isn't required that you understand how plug-ins or scripts work in conjunction with the IDE to produce a final binary program to ship to your customers. This manual is not relevant to you unless you are specifically interested in deriving new functionality from the IDE, through the creation of plug-ins or scripts that add some new functionality of your choosing.

In other words, reading this book is only required if you want to extend the CodeWarrior IDE to do new things beyond what it already does. Examples of this would be writing a script that told the IDE to compile all the source files, or building a template project from script commands, or creating a custom compiler that the IDE calls at compile time.

The sections in this chapter are:

- [Introduction to Extending the CodeWarrior IDE](#)
- [Read the Release Notes](#)
- [New Things](#)
- [Requirements for Developing Plug-ins](#)
- [What You Should Already Know](#)
- [Starting Points](#)
- [Other Resources](#)

# Introduction to Extending the CodeWarrior IDE

This manual accomplishes two objectives. First, it documents the CodeWarrior Plug-in APIs (Application Programming Interfaces), and shows you how to create compiler, linker, and other plug-ins to extend the capabilities of the CodeWarrior IDE. Second, it shows you how to get started writing scripts for extending the functionality of the CodeWarrior IDE, and for automating repetitive tasks.

An introduction to the terminology in use here in this manual is useful. First, you need to know what a plug-in is, and how that relates to the CodeWarrior IDE. You also need to know what scripting is, and why you might want to do it. This section includes the following topics:

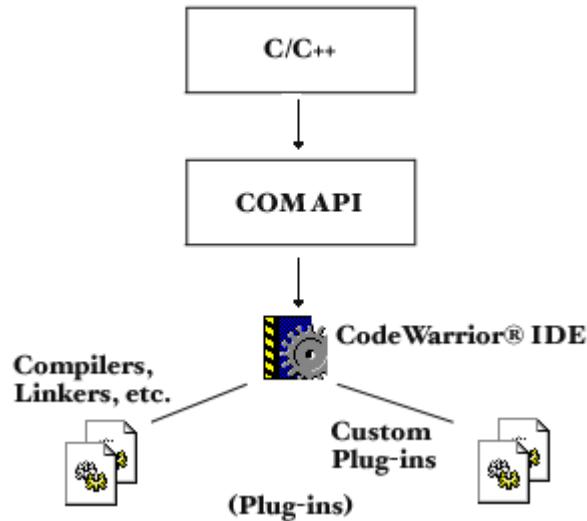
- [What is a Plug-in?](#)
- [What is a Script?](#)
- [In-Scripting, Out-Scripting, and Platform Differences](#)

## What is a Plug-in?

A plug-in is a separate piece of compiled executable code that the CodeWarrior IDE invokes at a specific point in time. Types of plug-ins include compilers, linkers, settings panels, plug-ins that connect the IDE to version control systems, and extensions to the CodeWarrior IDE, such as extensions to the text editor. You can actually build any arbitrary tool that does anything, but it looks like it's part of the IDE. You can access your project and target internals and work with them, or do things that have nothing to do with code. You could even write an astrology plug-in if you wanted to.

As shown in [Figure 1.1](#), you typically use C/C++ and Microsoft COM to create user interface plug-ins for the IDE. It is possible to use other means to drive the operations of the IDE. COM (Common Object Model) plug-ins extend the CodeWarrior IDE so you can add menus and toolbar items for performing other operations. You can also control plug-in-specific windows.

**Figure 1.1** Controlling the IDE from C/C++



The IDE invokes its resident plug-ins as necessary during use, without the user ever having to know how and when they're used. You create new custom plug-ins if you want to solve other problems for you. In other words, the CodeWarrior IDE makes use of standard plug-ins to do normal tasks, and you create new custom plug-ins if you need something special that the standard plug-ins do not provide.

The IDE calls on plug-ins without the plug-ins having to know the state of the IDE. For example, when the user chooses the IDE's **Make** command, the IDE—automatically and transparently to the user and to the plug-ins—coordinates the execution of its plug-in compilers and linkers to produce an executable program.

### What is a Script?

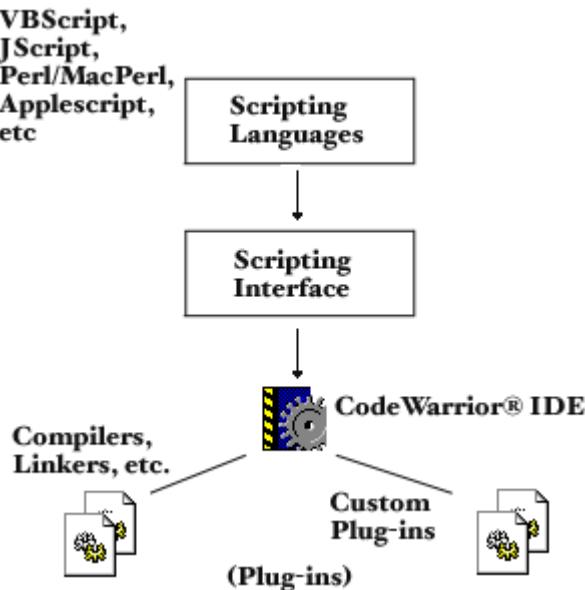
A script is a set of text commands in a file that you create, with the intention of commanding the CodeWarrior IDE to perform certain operations in a certain sequence.

One example of a script you might want to write is one that would check out the latest version of all your source code from revision control, build the code, and create the binaries for distributing your

end product. As this is a repetitive operation you perform often, creating a script to do this saves you time.

As shown in [Figure 1.2](#), you can write scripts that control the IDE in a variety of languages. You have many choices, including ECMA-compliant scripting languages like JScript, or JavaScript, or Perl and MacPerl, in addition to native languages like VBScript (Windows) and AppleScript (Mac OS). On Mac OS, the scripting language you choose interfaces with the IDE resources (aedt and aete resources) to provide the needed scripting functionality. On Windows, the scripting language you choose interfaces with the COM API that is exposed by the IDE.

**Figure 1.2** Scripting with the CodeWarrior IDE



### In-Scripting, Out-Scripting, and Platform Differences

To every general rule there is usually an exception, and the mechanisms you'll use to extend the functionality of the CodeWarrior IDE are no different. There are important platform-specific differences associated with extending the IDE on Mac OS, Windows, and UNIX® platforms, whether you are using scripted commands, programmatic control, or some other mechanism to accomplish your goals.

To clarify the concepts at hand, we'll introduce two new terms. In-scripting is the act of controlling the IDE by some external means. A script or an external program asks the IDE to perform some action, which it will then do. Since the action is inbound to the IDE, we say that the IDE is being in-scripted.

Out-scripting is the act of the IDE itself invoking some action as a course of its operations. By loading a custom linker plug-in, or calling a Perl script as a condition of building a target within the IDE, the outbound actions initiated from the IDE cause some external operation to execute.

Creating plug-ins using the plug-in API allows you to cause the IDE to invoke tightly-coupled operations that are all out-scripted by the IDE. The COM API's provide many interfaces allowing external parties to control the IDE (in-scripting) with opportunity to establish limited two-way interaction with the IDE by registering callbacks.

By use of a Perl plug-in, you can have the IDE generate arbitrary COM calls (out-scripting).

On Mac and UNIX both in-scripting and out-scripting using COM are limited to code that resides in plug-ins. The Metrowerks COM implementation operates only within a single process memory space (a so-called in-process implementation).

On the Mac, AppleScript supports both for in-scripted and out-scripted operations.

At this time, native scripting languages are usable on Mac and Windows, but there is not a native scripting methodology for the UNIX®-based CodeWarrior IDE.

The current command-line interface available on the UNIX IDE is quite limited and offers a single chance at issuing commands to the IDE. So think of this functionality as limited, fixed in-scripting.

The COM interfaces to the IDE are the same on all platforms. The real difference is in the tools available for dealing with COM interfaces. On Windows the native facilities require only the IDE interface definitions and employ the native COM initialization techniques.

## Overview

*Read the Release Notes*

---

plug-ins can contain any combination of plug-in API code, COM API (client) code, and COM server code. The Mac OS or UNIX IDE handle a COM server-only module as a plug-in since it has to be loaded into the IDE. On Windows, the COM infrastructure handles registering and locating COM servers, and the IDE does nothing special. Similarly, on Mac and UNIX, any COM client calls can only be from plug-ins. On Windows they can be from anywhere.

So as you learn from reading this section of the manual, the methods you use to extend the IDE will vary greatly between the different platforms supported by the CodeWarrior product family.

### More Information in This Manual

The sections in this chapter introduce you to other material you will find interesting depending on what you want to do. It isn't necessary to read this entire book before you begin creating plug-ins and scripts. You are encouraged to read the topics that are of importance to your task at hand.

- [Read the Release Notes](#)—getting last minute information about extending the CodeWarrior IDE
- [New Things](#)—improvements and changes in this documentation
- [Requirements for Developing Plug-ins](#)—what you'll need to develop plug-ins and scripts
- [What You Should Already Know](#)—what this manual assumes you know about using computers and computer programming
- [Starting Points](#)—how to use this manual
- [Other Resources](#)—information you'll find useful that isn't in this manual

## Read the Release Notes

Before referring to the rest of this manual, please read the release notes for the CodeWarrior IDE and other components. They contain important information about new features, bug fixes, and any late-breaking changes that arrived after this manual went to press.

# New Things

This document was revised to cover the latest versions of the different plug-in API you use to extend the CodeWarrior IDE.

---

**NOTE** Some features could not be documented fully before going to press.

---

## A Note About API Versions

The version of the API differs depending on which API you are working with. API version numbers are tied to the particular functionality of the plug-in you are creating. So the version number of the compiler API is different than the version number of the VCS (Version Control Systems) API.

For user interface panels, the latest version of the API is version 12. For compilers and linkers, the API version is 13. For VCS the current API version number is 9.

At press time, the plug-in APIs used in the current IDE have versions as show in .

**Table 1.1** **Plug-in API Version Numbers**

<b>API</b>	<b>Enumeration</b>	<b>Version</b>
Panels	DROPINPANELAPIVERSION_1 2	12
Compiler and Linker	DROPINCOMPILERLINKERAP IVERSION_13	13
VCS	VCS_API_VERSION_9	9

You determine whether an interface is present and its behavior by knowing what version of the API you are working with.

The COM API is not versioned separately from the IDE. Of course, part of the COM specification is that each interface has a version. The user can probe for the presence or absence of interfaces at particular versions since the version is part of the name.

## Overview

### Changes for Version 13

---

CodeWarrior Pro IDE versions 5 and later support compiler and linker plug-ins that use version 11 of the linker API. CodeWarrior IDE version 4.2.x (the Pro 6.2 release) and later support version 12 of the compiler and linker API.

## Changes for Version 13

---

**NOTE** Version 13 ships with CodeWarrior Pro8.

---

Changes in the version 13 APIs include:

- Support for long file name, which required a new structure for FileSpec. See [“CWFileSpec” on page 162](#).
- 

**NOTE** To get long filename support, your plug-in must link to PluginLib5.

---

- New compiler/linker methods to support frameworks: [CWGetFrameworkCount](#), [CWGetFrameworkInfo](#), and [CWGetFrameworkSharedLibrary](#). These threee methods have an associated data structure: [CWFrameworkInfo](#).
- A new plug-in method to improve managing files within a project: [CWRemoveProjectEntry](#)
- A new structure for [PanelParameterBlock](#).
- A new linker flag: [linkerSuggestsNonRecursiveAccessPaths](#). This flag can save time and resources by making simpler paths.
- Two new compiler flags: [kCompReentrant](#) and [kCompRequiresTargetCompileStartedMsg](#). kCompReentrant supports concurrent compilation, while kCompRequiresTargetCompileStartedMsg supports pre- and post-processing.

## Changes for Version 12

Changes in the version 12 APIs include:

- [New Compiler Plug-in Flag](#)
- [Widespread DEF File Bug Fix](#)

## New Compiler Plug-in Flag

In version 12, a compiler flag was added, and also two new requests to support notifying compiler plug-ins of the start and end of a target build compile step. This first use of this feature is likely to be Metrowerks compilers intending to generate object code taking advantage of optimization involving inter-module knowledge. You may find another use for it in your plug-in creation forays.

The compiler drop-in capability flag is:

```
kCompRequiresTargetCompileStartedMsg
```

and indicates that the compiler plug-in is interested in being called at the start and at the end of the compile step during the build of a target. The start notification is made before the first compile (if any) in the target and the end notification is made after the last compile in the target finishes.

Starting in version 12, the sequencing of the notifications fits in with the build sequence as shown here:

- Project/SubProject Build Start
- Target Build Start
- Target Compile Start - notification sent prior to this phase.
- Target Compile End - notification sent after this phase.
- Target Link Start
- Target Link End
- Target Build End
- Project/SubProject Build End

The two new requests sent to compiler plug-ins to deliver the notifications are:

```
reqTargetCompileStarted  
reqTargetCompileEnded
```

## Widespread DEF File Bug Fix

Be aware of an issue plug-in developers may run into when updating to the latest x86 linker. This is the result of a bug fix which may expose the incorrect use of export files.

When exporting plug-in callback symbols (`main`, `CWPlugin_GetDropInFlags`, etc) via a linker definition file (`.def`) or linker command file (`.cmd`), be sure that you are not using the keyword `NONAME`. This keyword tells the linker not to make symbol names visible from the DLL binary. Unfortunately, this keyword was incorrectly implemented by the linker until recently. If `NONAME` is being used, DLLs built with the new linker will no longer load into the Windows IDE.

The IDE does symbol-only lookups on callbacks exported from DLLs, so the use of enumerations (@<number>) is unnecessary. `NONAME` is used in combination with enumerations to hide the symbol name so that only the numerical index is available to obtain an export's address.

The sample plug-ins in the Codewarrior plug-in SDK had included `*.def` files which inappropriately assigned enumerations to symbols and exported them with the `NONAME` keyword.

As it is likely that a large number of Codewarrior developers have copied these `*.def` files for use in their own projects, be aware of this issue.

## Changes in Version 11

Changes in the Version 11 API include:

- A change in the behavior of [CWGetFileInfo](#). The `objmoddate` of the [CWProjectFileInfo](#) structure returned now reports the date of last object code update, where previously it returned the modification date of the source file.
- The [CWGetProjectFile](#) callback now always returns the file spec of the actual project file. Previously, for compilers it would return the file spec of the target data file.
- A new [linkerGetTargetInfoThreadSafe](#) flag may be returned by the `CWPlugin_GetDropInFlags` entry point. This should be set by a linker if it handles the [reqTargetInfo](#) request in a thread-safe way (without reentrancy problems).
- There is a new callback named [CWResolveRelativePath](#). It resolves a [CWRelativePath](#) structure to a [CWFfileSpec](#).
- There are several new callbacks to support access to settings in the **Runtime Settings** panel: [CWGetCommandLineArgs](#),

[CWGetWorkingDirectory](#), [CWGetEnvironmentVariable](#), and [CWGetEnvironmentVariableCount](#). These are useful when adapting command line tools to the CodeWarrior IDE.

- A new [CWPanelChooseRelativePath](#) callback displays a dialog box to allow the user to choose a relative path.
- New callbacks exist to support the Pro 5 feature of exporting a project to an XML file and then reimporting it. The new callbacks convert the opaque panel data handle to and from a set of name-value pairs that are represented in XML.
- There are two new requests, [reqWriteSettings](#) and [reqReadSettings](#), that are sent to the panel when the project is being exported and imported.

## Changes in Version 8

Changes to the version 8 version control API for the CodeWarrior Pro 5 product include:

- The date field of the `CWVCSVersionData` was changed from being a C standard library `tm` structure to being a [CWFfileTime](#).
- There is a new callback called [CWVCSStateChanged](#), which replaces the [CWFfileStateChanged](#) callback from the version 7 API, to make the API more language neutral.
- The `CWVCSItem` struct now includes the checkout state of the item. This field needs to be set by the plug-in before calling [CWSetVCSItem](#).

In addition, other changes have been made to this document, including:

- Reorganized the discussion material into chapters on general issues, compiler and linker creation, and panel creation.
- Added a short chapter on development setup.
- Numerous corrections to reference material, and general cleanup and editing.

## Requirements for Developing Plug-ins

These topics describe the items you'll need to develop plug-ins for the CodeWarrior IDE:

## **Overview**

*Installing CodeWarrior Software*

---

- [Installing CodeWarrior Software](#)
- [Installing the Plug-in API SDK](#)

## **Installing CodeWarrior Software**

To create plug-ins or scripts for the CodeWarrior IDE, you'll need a CodeWarrior product package that comes with the tools and files needed to develop software for the operating system or computer platform on which your plug-ins will run.

To test your plug-in or script, you'll need a CodeWarrior package that includes a version of the CodeWarrior IDE that runs on the same operating system or platform on which your plug-in will run.

Typically, you'll write and test your plug-ins or scripts on the same computer and operating system that you're developing the plug-in on. More detail on configuration your environment for plug-in development is covered in [“Overview” on page 39](#).

To install the CodeWarrior software, follow the instructions in the QuickStart guide of your CodeWarrior product.

## **Installing the Plug-in API SDK**

To develop CodeWarrior IDE plug-ins, you should start by obtaining and installing the plug-in SDK. Read [“Overview” on page 39](#) to continue the SDK setup process.

## **What You Should Already Know**

This manual shows you how to create a plug-in or script for use with the CodeWarrior IDE.

What this manual shows you:

- what a plug-in is and what types of plug-ins there are
- how plug-ins interact with the CodeWarrior IDE
- how to create the parts of a plug-in that interact with the IDE using the C/C++ programming languages
- how to write a script that controls the IDE

What this manual doesn't show you:

- the design and implementation of compilers, linkers, revision control systems, or text editors
- developing software for the platform on which your plug-in will run
- how to use the CodeWarrior IDE
- how to use other CodeWarrior tools to develop software

## Starting Points

Writing a CodeWarrior IDE plug-in or script isn't difficult. However, you need to know where to go in the documentation to find the information that you need.

### Getting started

To get started developing a CodeWarrior IDE plug-in, see the following:

- For an overview of plug-ins and the plug-in development process, see [“Overview” on page 53](#).
- For essential information about the process of developing plug-ins using CodeWarrior, see [“Plug-in Development Setup” on page 39](#).
- For a discussion of general information related to developing all types of plug-ins, see [“Creating CodeWarrior IDE Plug-ins” on page 57](#).
- For general reference material on all plug-ins, see [“Plug-in API Reference” on page 99](#).
- For a discussion of compiler and linker plug-ins, see [“Creating Compiler and Linker Plug-ins” on page 199](#).
- For reference material on compiler and linker plug-ins, see [“Compiler and Linker Plug-in Reference” on page 237](#).
- For a discussion of the preference panel API, see either [“Creating Settings Panel Plug-ins on Windows” on page 375](#) or [“Creating Settings Panel Plug-ins on Mac OS” on page 383](#).
- For reference material on the preference panel API, see [“Settings Panel Plug-in API Reference” on page 399](#).

- For detailed information about the format of the data used by the IDE's symbol browser, see [“Browser Reference” on page 573](#).
- For detailed information on the Mac OS object and library code formats used by CodeWarrior, see [“PowerPC Object Code \(Mac OS\)” on page 593](#) and [“Library Specification \(Mac OS\)” on page 627](#).
- To add support for the IDE to a separate text editor, see [“External Editors on Mac OS Overview” on page 631](#).
- For information on developing software that isn't covered in this manual, see [“Other Resources” on page 38](#).

To get started writing scripts that control the IDE, refer to the following:

- For information on scripting in a Mac OS environment, refer to [“CodeWarrior Mac OS Scripting Overview” on page 655](#).
- For information on scripting in a Windows environment, refer to [“CodeWarrior Windows Scripting Overview” on page 731](#).

## Other Resources

For other information on using the CodeWarrior IDE to develop software for the platform on which your plug-in will run, see the applicable *Targeting* manual. For example, to learn about using the CodeWarrior IDE for targeting the Win32 platform (Windows 95/NT), see the *Targeting Windows* manual. For Mac OS, see *Targeting Mac OS*.

For general information on using the CodeWarrior IDE, see the *IDE User Guide*.

# Plug-in Development Setup

---

This chapter describes how to set up your development environment to create CodeWarrior plug-ins.

## Overview

Metrowerks distributes a software development kit (SDK) for developing CodeWarrior IDE plug-ins. This short but important chapter describes setting up the plug-in SDK and the mechanics of the plug-in development process. All plug-in developers should read this chapter.

This chapter covers the following topics:

- [About the SDK](#)
- [Getting Started](#)
- [Plug-in Installation](#)
- [Plug-in Debugging](#)
- [Troubleshooting](#)

## About the SDK

This section briefly describes the plug-in SDK and where to find sample scripts:

- [SDK Overview](#)
- [SDK Contents](#)
- [Resources for Scripting](#)

## SDK Overview

Most likely, you received this package as part of the SDK, which ships on the CodeWarrior CDs. You can find the latest version of the complete SDK on the Metrowerks web site:

---

<http://www.metrowerks.com/support/api/>

---

**TIP** This SDK is continually updated, typically after major CodeWarrior releases. Be sure to check the web site periodically for updates.

---

The SDK includes sample code illustrating each of the plug-in types supported by the IDE, including: a compiler, a linker, a preference panel, and version control plug-in samples. The samples provide limited practical functionality, but illustrate the essential steps you will need for constructing various plug-in types.

All the SDK example projects are cross-platform except the preference panels which are platform-specific due to major differences in the panel API.

## SDK Contents

The SDK includes the following components:

- Sample projects, one for each major plug-in type (plus a preprocessor, which is a specialized compiler).
- A “build all” project, useful as a single starting point for accessing other sample projects, and for rebuilding all samples.
- Header files, which define the CodeWarrior IDE plug-in interface, and the routines, data types, and constants it exports publicly.
- Library files, for linking into your plug-in projects. These libraries provide glue and implementation code for the plug-in API routines.
- This documentation, describing the APIs and how to use them.

## Resources for Scripting

The CodeWarrior IDE can be controlled by scripts, which can be written in AppleScript (for the Mac OS), VBScript (for Windows) or Perl (for Mac OS and Windows).

You can find sample scripts that illustrate techniques for controlling the CodeWarrior IDE are on the CodeWarrior CDs. The scripts are for use with AppleScript (Mac OS) or VBScript and the Windows Script Host (Windows).

The following chapters describe how to write scripts for controlling the IDE:

- [Mac OS CodeWarrior Scripting](#)
- [CodeWarrior Scripting on Microsoft Windows](#)
- [Perl Scripting](#)

## Getting Started

The following steps describe how to set up the SDK on your machine for a trouble-free development experience. If you deviate from these steps, you may encounter problems (see ["Troubleshooting" on page 51](#)).

### Installation Steps

Follow these steps to set up the CodeWarrior IDE Plug-in SDK example code on your computer:

**1.** Copy files to hard disk

Locate the SDK on your CodeWarrior install CDs, and copy its contents to any desired location on your local hard disk. If the SDK did not come on your CodeWarrior CDs, you can download it from the Metrowerks web site:

`http://www.metrowerks.com/support/api/`

**2.** Open Build All project

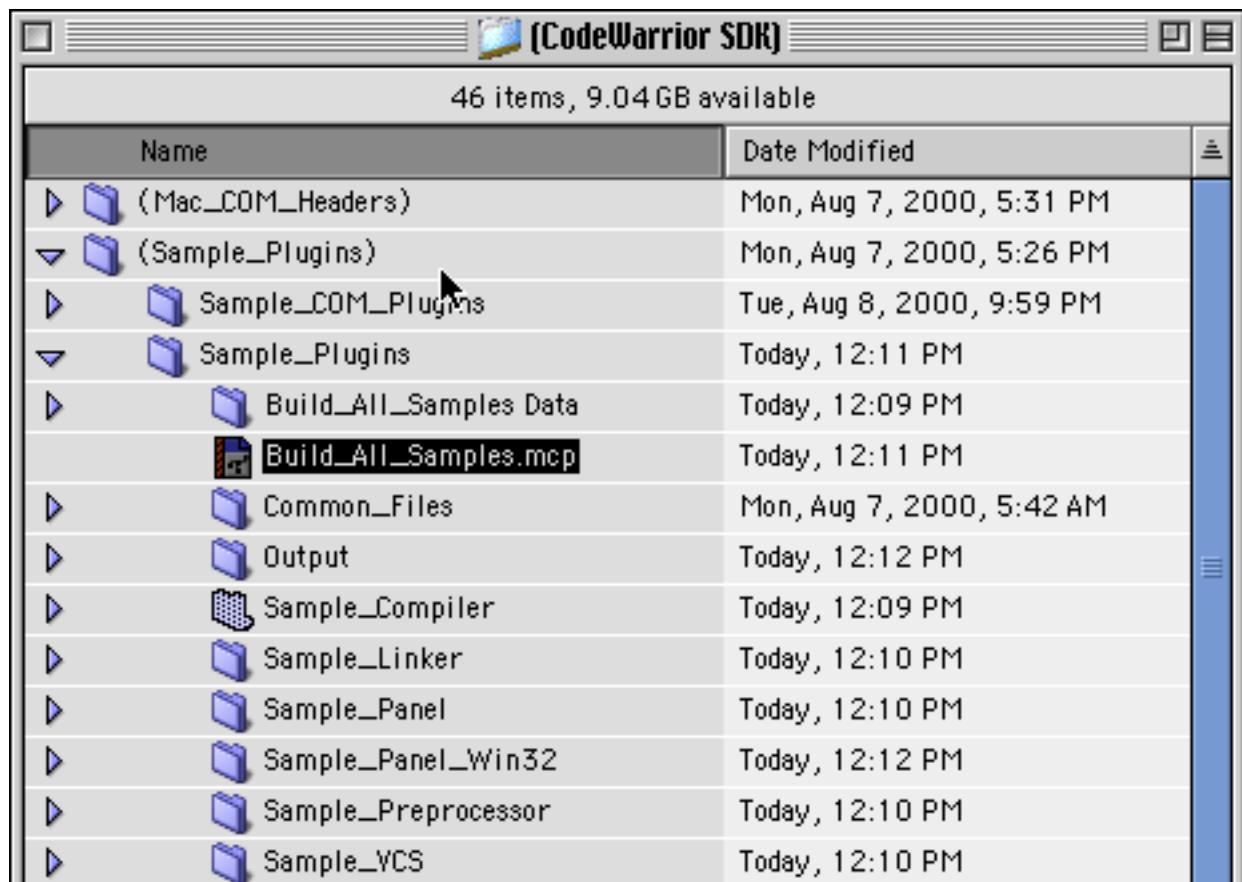
The plug-in SDK includes a ‘Build\_All\_Samples.mcp’ project, as shown in [Figure 2.1](#) (Mac OS) and [Figure 2.2](#) (Windows). In

## Plug-in Development Setup

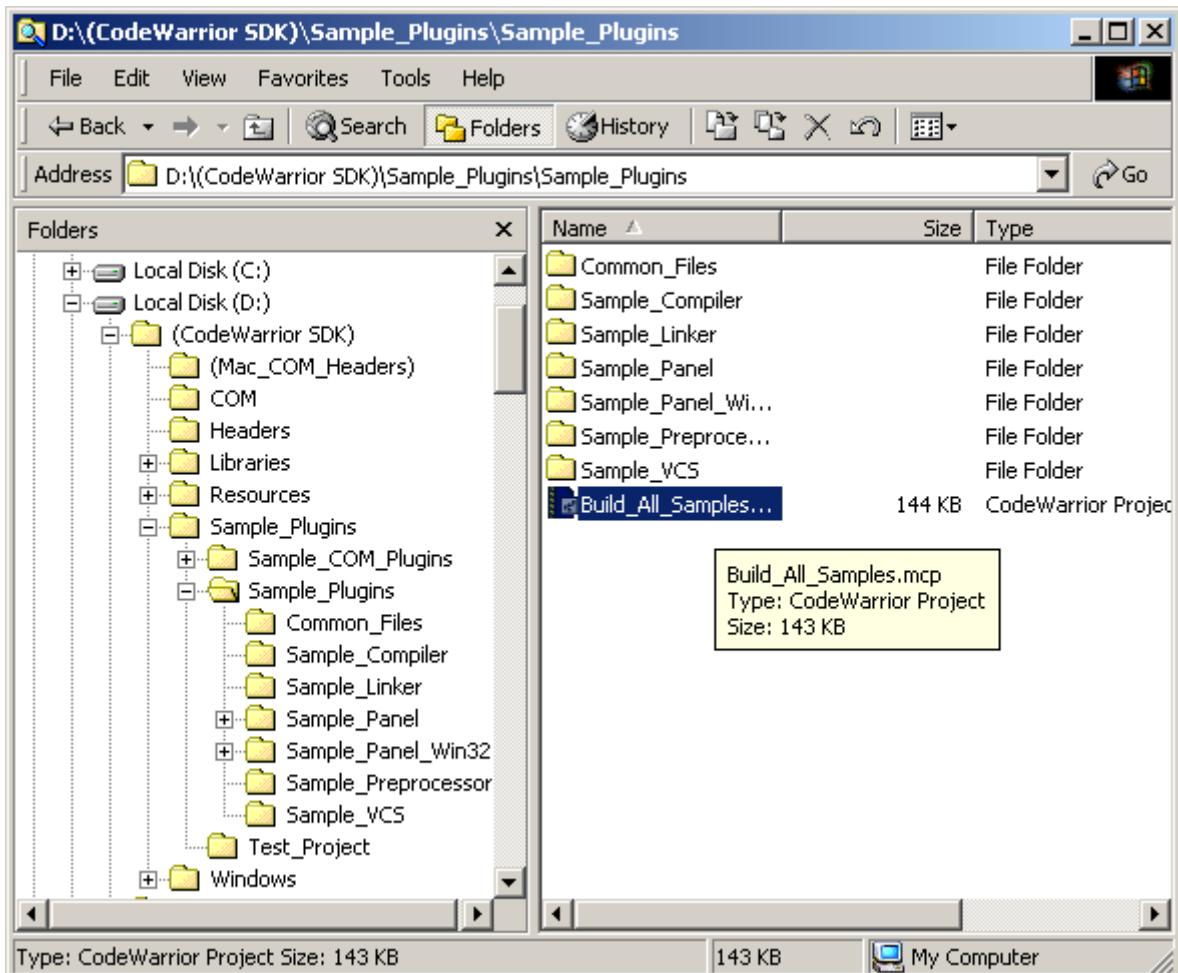
### Installation Steps

In addition to the sample projects, a copy of this build-all project may be useful as a single starting point for building your own plug-ins, and for rebuilding all samples.

**Figure 2.1 Location of Build\_All\_Samples.mcp (Mac OS)**



**Figure 2.2 Location of Build\_All\_Samples.mcp (Windows)**



### 3. Verify project output paths

For each project in the ‘Build\_All\_Samples.mcp’ project, open the project file, and click the **Target Settings** button at the top of each opened project window. Switch to the **Target Settings** panel and verify that the **Output Directory** is set up properly. The output directory should be set to an appropriate subfolder of the CodeWarrior plug-ins folder as described in [“Plug-in Installation” on page 44](#). You may wish to create a new subfolder, if you prefer to keep development versions of your plug-ins separate from the plug-ins that came with the stock CodeWarrior software environment.

### 4. Build all plug-in sample projects

Once all output directories have been specified correctly, click on the make icon in the ‘Build\_All\_Samples.mcp’ project window.

Another way to rebuild all samples is open the ‘Build\_All\_Samples.mcp’ project, and choose **Make** on the **Project** menu. The CodeWarrior IDE examines subprojects included in this project, and recompiles any that are not up to date. This is easier than opening and compiling each project individually.

**5.** Quit the IDE and relaunch it

To force the IDE to notice the newly created and installed plug-ins, quit the IDE, and restart it.

**6.** Open and make SampleProject.mcp

In a sibling folder to the one containing all the sample projects, there should be a ‘TestPlugins’ folder containing a ‘TestPlugins.mcp’ project. Open this project, and open **Target Settings**. If the IDE issues an error when attempting to open **Target Settings**, it probably means that one or more of the plug-ins was not compiled and installed correctly. Otherwise, it should be possible to switch to the target settings panel, and verify that the chosen linker is **Sample Linker**. Also, a preference panel named **Sample** should appear in the list of preference panels.

If you’ve made it this far, the SDK is set up and ready for use. You can browse the projects as needed and get acquainted with the code. Read the rest of this chapter to find out more about the plug-in development process.

## Plug-in Installation

Before plug-ins can be tested, they must be installed. This section explains how to install plug-ins:

- [Plug-ins Folder](#)
- [Automatic Installation](#)

### Plug-ins Folder

To test plug-ins, they must be placed in the appropriate location on your hard disk to be recognized by the IDE.

Windows	On Windows, plug-ins must be placed inside one of the directories along the following path:
---------	---

---

<Install Folder>\Metrowerks\CodeWarrior\Bin\Plugins

---

Note that C:\Program Files is the typical location that the tools are installed to by default, but your installation may vary.

- Mac OS      On Mac OS, plug-ins must be placed inside one of the directories along the following path:

---

<Install Folder>:Metrowerks CodeWarrior:CodeWarrior Plugins

---

Plug-ins should usually be placed in subfolders whose names match the plug-in types. For example, place compilers and assemblers in the ‘Compiler’ subfolder of the Plugins folder. Place linkers, prelinkers, and postlinkers in the ‘Linkers’ subfolder.

Plug-ins can in fact be placed anywhere within the Plugins directory and still be recognized by the IDE. This could be useful if you prefer to keep developmental plug-ins separate from standard Metrowerks plug-ins.

## Automatic Installation

Plug-ins can be installed manually, or by configuring the CodeWarrior IDE to place the compiled plug-ins in the appropriate Plugin folder automatically. To configure the output directory for a compiled plug-in, open **Edit > Target Settings**, and choose a folder in the box labeled **Output Directory**.

- 
- NOTE**      The IDE will not recognize newly installed plug-ins unless you quit and restart the IDE. In general, this will happen automatically if you use the debugging setup described in “[Plug-in Debugging” on page 46](#). This is because the second copy of the IDE used to debug plug-ins will be quit and restarted each time you rebuild and debug a plug-in. This ensures that the second IDE’s knowledge of installed plug-ins is up to date.
-

# Plug-in Debugging

Debugging plug-ins requires special setup and configuration of the CodeWarrior IDE. In addition, the IDE offers several debugging aids specific to developing plug-ins. Read this section to learn how to set up the IDE to debug plug-ins.

You will typically repeat many of these steps each time you set up a new project for development. However, you will only duplicate the IDE (step 1) once.

## Debugging Setup

The process of debugging a plug-in requires that it be running inside the CodeWarrior IDE. Since most debuggers, the Metrowerks debugger included, can debug other application processes but not themselves, it is necessary to create a copy of the IDE that serves as the host application for running and testing plug-ins, and the application that will be targeted by the debugger during debug sessions. We'll refer to the IDE that is the host as the Master, and the copy of the IDE that loads the plug-ins for debugging will be called the Slave.

Follow these steps to set up your development environment properly for plug-in debugging:

1. Locate and duplicate the Codewarrior IDE executable

Windows On Windows, the IDE is named `IDE.EXE` and is usually found in:

`Program Files\Metrowerks\CodeWarrior\Bin`

Mac OS On Mac OS, the IDE application is usually found in:

`<Install Folder>:Metrowerks CodeWarrior`

Regardless of platform, duplicate the IDE executable (click then Command-D in the Mac OS, or in Windows right-click **Copy**, right-click **Paste**). You may use any name you like for the copy, but naming it something memorable may assist you in distinguishing the copies.

2. Specify the CodeWarrior IDE as the execution host

Open a plug-in project that you intend to debug with the Master copy of the IDE. If you are just starting plug-in development, you will probably want to duplicate one of the starter projects to open, or simply experiment with an original. Once open, use **Edit > Target Settings** (Target Settings will actually take the name of the target instead) and choose the **Runtime Settings** panel. In the section titled **Host Application for Libraries & Code Resources**, click the **Choose** button and locate the copy of the IDE created in step 1 (the renamed copy), then click the **Save** button.

3. Enable automatic library targeting

Using **Edit > Target Settings**, choose **Debugger Settings**, and in the section titled **Other Settings**, enable the checkbox labeled **Auto-target libraries**. This ensures that the CodeWarrior debugger will notice plug-ins being loaded by the Slave copy of the IDE.

4. Configure the output folder

This step is not absolutely essential, but will save time and avoid possible confusion. In **Edit > Target Settings**, select the **Target Settings** panel. In the section titled **Output Directory**, click the **Choose** button, and select the appropriate CodeWarrior plug-ins subfolder, as described in [“Plug-in Installation” on page 44](#). This setting tells CodeWarrior to place compiled plug-ins in its plug-in folder.

---

**NOTE** Caution, this overwrites any existing similarly-named plug-in in the same location at the completion of the build!

---

5. Enable Debugger Launching

Choose **Enable Debugger** on the **Project** menu in the main copy of the IDE so it will launch the other copy of the IDE at the completion of the build.

You are now set up to run and debug plug-ins like any other piece of code. When you select **Project > Run**, CodeWarrior will build your plug-in, create a compiled library in the appropriate plug-in folder, and launch the second copy of the IDE to load the plug-ins you just built. During startup, the second IDE copy will scan for plug-ins, and interrogate their capabilities. When required by the

actions taken in the second IDE, the IDE will call your plug-in to perform needed services, such as compiling or linking.

The debugger running in the first copy of the IDE will detect when execution in the second copy has entered your plug-in, and reached a breakpoint placed in the plug-in source. This will bring the first copy of the IDE frontmost, for code inspection and debugging.

---

**NOTE** Debugging one copy of the IDE using another can be a confusing experience. One way to help keep track of the copies is to configure each copy differently in some significant visual way, such as by arranging windows differently in each.

---

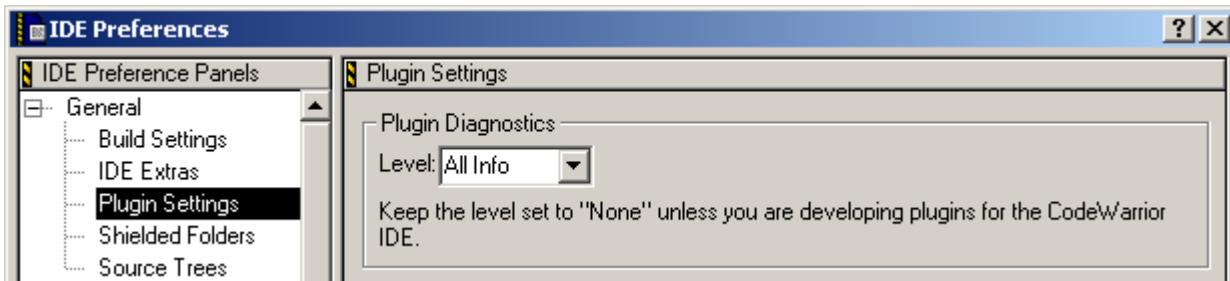
## Plug-in Debugging Options

In addition to the basic debugging setup outlined in [“Debugging Setup” on page 46](#), you may find some other IDE options useful while debugging plug-ins. Which options and when they prove useful will vary with the plug-in type and the stage of plug-in development.

### Plug-in Diagnostics

To enable plug-in diagnostics, choose **Edit > Preferences** and under the **Plugin Settings** panel, choose either **Errors Only** or **All Info** from the popup menu labeled **Level**, as shown in [Figure 2.3](#). This setting applies only to CodeWarrior plug-ins and controls the level of informational messages presented by the IDE when loading and communicating with plug-ins. You probably want to set this only in the Slave IDE.

**Figure 2.3** **Plug-in Settings**



When set to **Errors Only**, the IDE issues message only for significant errors in plug-in operation. When set to **All Info**, the IDE output includes messages about possible incorrect or undesirable behavior, obsolete plug-in calls, and status messages. When **All Info** is enabled, the IDE also issues a plug-in report summarizing all known information about installed plug-ins, including associations among plug-ins.

Because the volume of information returned by the **All Info** setting can be a bit overwhelming, especially for routine use, you may wish to use the **Errors Only** setting most of the time. You can switch to **All Info** occasionally to check the details of plug-in compliance with API standards, or to get more information in particularly difficult debugging situations. You should probably run with **Level** set to **All Info** at least once prior to shipping your plug-in.

### **Browser Symbol Dump**

To enable dumping of browser symbol information, select **Edit > Target Settings** and under the **Build Extras** panel, enable both the **Activate Browser** and **Dump internal browse information after compile** checkboxes.

This setting generates a textual representation of the browser information generated while compiling a file. This is useful for debugging errors in the browser symbol streams returned by a compiler plug-in.

---

**NOTE** To generate a symbol dump, use the **Project > Compile** menu command. Symbols are not generated during a **Project > Make** or **Project > Build** operation. The only way to generate symbol information for more than one file is to select multiple files in the project window before compiling. This is rarely desirable since usually even a single file will generate large amounts of browser data debugging information.

---

### **Listing 2.1    Sample Browser Dump**

---

```
-- Sample_Compiler.c --
```

```
Local file IDs:
```

## Plug-in Development Setup

### Plug-in Debugging Options

---

```
[1] 'D:\CW_Plugin_SDK\(\Samples)\Sample_Plugins\Sample_Compiler\  
      Sample_Compiler.c' [size: 7917 ]  
  
<rest of file list omitted>  
  
-- Browse Header --  
header: 0xBEABBAEB  
version: 2  
language: C (1)  
uses name table: true (1)  
earliest compatible version: 2  
reserved: (60 bytes)  
-----  
  
[76]  
item: global (1)  
contrib file: 1 src file: 1  
start/end offset: 870:893  
reserved: (4 bytes)  
simple name = length: -1, nametable ID: 1 --> "linecount"  
qualified name = length: 0  
flags: static (0x0002)
```

---

[Listing 2.1](#) illustrates sample browser dump output. Browser information starts with a file list (truncated here for brevity), and continues with a header, which is mostly straightforward. Some fields of the output are followed by numbers in parentheses which indicate the numerical value corresponding to a preceding textual description. For example, in the sample above, a numerical code of 1 corresponds to the 'C' programming language.

The header is followed by one record per browser symbol generated while compiling a file. This may include symbols defined in other included files. Additional entries for structure and class members appear indented within their containing objects.

Each browser record includes the following information:

- its stream offset (zero-based, from start of stream), in bytes, in brackets
- the symbol type (global, function, typedef, class, etc.)

- numerical source file indexes, which refer to files listed at the start of a symbol dump
- the character offsets for the start and end of the symbol definition within its source file
- the simple and qualified names of the symbol
- various symbol property flags
- additional information specific to the type of symbol

For more information about browser data, see [“Browser Reference” on page 573.](#)

### Debug Message Logging

Another IDE feature that may prove useful is CodeWarrior’s automatic logging of error and debugging messages. To enable message logging, select **Edit > Target Settings**, and under **Debugger Settings**, enable the checkbox labeled “Log System Messages.”

Windows	On Windows, this option records all debugging <code>printf</code> statements, as well as messages about DLL loading and unloading.
Mac OS	On Mac OS, this option records all <code>DebugStr</code> messages, as well as messages about shared libraries loading and unloading.

## Troubleshooting

This section lists some possible problems you may encounter in setting up the SDK, especially if you do not follow the steps listed in [“Getting Started.”](#) You may also encounter these problems later, when modifying the SDK samples to suit your purposes, or when developing your own plug-ins from scratch.

### Cannot Add Files to a Project

If you find that the IDE will not let you add files having a particular extension or file type to a test project, check the following:

- Verify that the proper linker for your tool’s target OS and CPU is selected in the **Target Settings** project preference panel.
- If you are developing a linker, it may not be reporting default file mappings correctly. See [“Specifying File Mappings.”](#)

- If you are developing a compiler, it may be specifying the wrong target CPU and OS. In this case, the compiler CPU and OS will not match the linker's, and the IDE will not add the compiler's file mappings to the project. Again, see "[Specifying File Mappings.](#)"

### Cannot Select Compiler

If you have selected the correct linker in **Target Settings**, but you do not see your compiler in the **Compiler** popup menu on the **File Mappings** panel, this may be caused by one of the following:

- The linker and the compiler specify mismatching CPU or OS codes (at least one of which is wrong).

### IDE Displays ‘Error #2’ Message

The IDE displays this message when you attempt to select a plug-in for use that requires additional plug-ins that cannot be found. To fix this problem, compile the additional necessary plug-ins, and install them in the CodeWarrior plug-ins folder. Be sure to quit and restart the IDE. It should then be possible to select the compiler or linker successfully.

### Plug-in Behavior Incorrect or Debugger Not Stopping in Source

If you know you have installed your plug-in correctly, but it seems to be exhibiting incorrect behavior, or the IDE completely ignores any breakpoints you set in your source, check the following:

- Verify that there is only one copy of each plug-in installed. If you have multiple plug-ins that return the same OS and CPU information to the IDE, the IDE will fail to load all but one of them. This can be very confusing; it may appear that your plug-in is enabled, when in fact another is executing.

To assist in checking for this, you can use [Plug-in Diagnostics](#), which will report duplicate plug-ins.

# IDE and Plug-in Architecture

---

This chapter describes the kinds of plug-ins the IDE uses and describes the plug-in development process.

## Overview

The CodeWarrior IDE is a stand-alone application that provides software development services to a programmer. Among the services that the IDE provides are source code editing and browsing, compiling, linking, and project and file management.

Some of these services that the IDE provides are actually supplied by its plug-ins. A plug-in is a separate piece of software that the IDE calls on to perform compiling, linking, version control, or source code editing tasks for the user. The IDE decides which plug-ins to call transparently to the user, based on project settings and preferences that the user sets in the IDE.

There are several types of plug-ins that the IDE supports, and with few exceptions, they are supported on all IDE host platforms:

- Compiler and linker plug-ins provide source code translation, preprocessing, precompilation, disassembly, linking, and other services.
- Version control plug-ins integrate source database management into the IDE.
- Preference plug-ins extend the IDE's project settings dialog to support other plug-ins.
- Command line tool support is currently provided only for Windows, by way of sample code that can be adapted to other command line tools.

- External editor support is currently provided only for Mac OS.

Most of the CodeWarrior Plug-in API is uniform across platforms. There are differences in some parts of the API (primarily preference panels), but in general the API facilitates easy cross-platform development from a single source code base. Most plug-ins can be developed for all host platforms with modest additional effort beyond that required to support one platform.

## Types of Plug-ins

This section describes the kinds of plug-ins that are documented in this manual. These plug-ins are:

- [Compiler, Pre-linker, Linker, and Post-linker Plug-ins](#)
- [Settings Panel Plug-ins](#)
- [Version Control Plug-ins](#)
- [Third Party Editors \(Mac OS\)](#)

### **Compiler, Pre-linker, Linker, and Post-linker Plug-ins**

Compilers and linkers in most software development environments are used to perform discrete, well-defined tasks. A typical compiler translates information from one format, typically a source code file, to another format, typically an object code file. A typical linker resolves references and combines object code files into an executable program, static library, or other final binary file.

The CodeWarrior IDE supports plug-ins providing such conventional services, but it also affords plug-ins considerable flexibility in the actions they take in response to requests from the IDE. This design creates many opportunities for more flexible software development.

Some possibilities supported by this design include:

- Compilers that translate source files from one language to another, and add the output files to the current project, for subsequent compilation by another compiler.

- Compiler plug-ins that translate object code from one format to another making it possible to support libraries from other development environments.
- Pre-linkers (plug-ins that the IDE calls before the traditional link stage of the build process) that process compiler object data before the linker processes it.
- Linker plug-ins that examine source code to gather information about how to link a program.
- Post linkers (plug-ins that the IDE calls after the traditional link stage) that reprocess a file after it has been linked by a traditional linker (so that, for example, the same executable code can be reformatted to run on different platforms).

To create a compiler, pre-linker, linker, or post-linker plug-in, refer to [“Getting started” on page 37](#).

## **Settings Panel Plug-ins**

A settings panel plug-in provides a user interface to a compiler or linker plug-in’s options. A settings panel appears in the IDE’s Target Settings dialog box. Settings panel plug-ins (also known as preference plug-ins) store their settings with the current project. While performing an operation, a compiler or linker plug-in may retrieve a settings panel’s data.

See [“Creating Settings Panel Plug-ins” on page 331](#) for more information about preference panel plug-ins.

## **Version Control Plug-ins**

Version control system (VCS) plug-ins connect the IDE to source management systems and source code databases. They support checking source code (and other files) in and out of projects, adding files to projects, version comparison and history examination, and database browsing and inspection.

See [“Version Control System Plug-in API” on page 523](#) for more information about version control system plug-ins.

## **Third Party Editors (Mac OS)**

An editor application that is separate from the IDE can be enhanced to communicate with the IDE. The IDE accepts Apple Events from and sends Apple Events to an external editor to handle opening text files and marking modified and opened files.

To add support for the IDE to an external editor application, see [“External Editors on Mac OS Overview” on page 631](#).

## **COM Servers (Mac and UNIX OS)**

For the Mac and UNIX IDEs, COM servers are treated as plug-ins, because they have to be loaded into the IDE to make them usable.

# Creating CodeWarrior IDE Plug-ins

---

This chapter discusses basic concepts important to developing all types of plug-ins for the CodeWarrior IDE.

## Overview

This chapter explains the plug-in types and the basics of plug-in interaction with the CodeWarrior IDE, and it outlines services and concepts common to all IDE plug-ins.

This chapter covers the following topics:

- [Plug-in Binary Formats and Installation](#)
- [Platform Differences in Plug-in API](#)
- [Getting Runtime Information](#)
- [Informational Entry Points](#)
- [Responding to IDE Requests](#)
- [Managing Memory](#)
- [Managing Files](#)
- [Managing Plug-in Data](#)
- [Handling User Interaction](#)
- [Error Handling](#)

## Plug-in Binary Formats and Installation

This section describes the binary format of plug-ins, which vary somewhat by platform.

## Plug-in Binary Formats

On all platforms, plug-ins are dynamically loadable pieces of code that export required entry points, which the IDE looks for when loading plug-ins. The specifics of plug-in binaries differ according to platform and plug-in type.

### Entry Points

The `CWPlugins.h` header file provided in the plug-in SDK defines a macro `CWPlugin_ENTRY`. This macro should be applied to all C functions exported by a plug-in. This macro ensures that the entry point follows the appropriate calling conventions on the target platform. On Windows, entry points must use `__stdcall` calling conventions, and on Mac OS must use `pascal` calling conventions. All entry points return a result of type `short`.

### Windows Plug-in Binaries

The CodeWarrior IDE runs on PCs running Pentium-class or better processors running the Windows 95, Windows 98, and Windows NT (or later) operating systems.

The plug-in binary format for the Windows hosted IDE is a standard Windows 32-bit DLLs (dynamic-link library). Plug-in DLLs export the entry points required by the IDE by ordinal.

### Macintosh Plug-in Binaries

The CodeWarrior IDE runs on Mac OS computers that use either Motorola 680x0 (68K) or PowerPC processors. The IDE is a “fat” application, meaning that it is a single binary that executes on either processor. As a consequence, its plug-ins should also be fat.

A fat plug-in contains two versions of its executable code to support both 68K and PowerPC processors:

- a PowerPC CFM (Code Fragment Manager) shared library for PowerPC code
- a 68K CFM shared library or 68K code resources for 68K code

When both PowerPC and 68K use CFM, the code fragments for both processors are normally placed in the data fork of the plug-in

binary, using CodeWarrior's Mac OS Merge linker to combine PowerPC and 68K shared libraries into one fat library.

**NOTE** On MacOS X, plug-ins are restricted to PPC.

---

### Macintosh PowerPC

PowerPC plug-in code is stored in a CFM (Code Fragment Manager) shared library, in the data fork of a plug-in file. All required PPC entry points are exported by name, and must match those used in the sample projects.

### Macintosh 68K

Macintosh 68K code is more complicated. There are two options for packaging 68K code: it may be provided as a collection of A4-based code resources, or as a code fragment under CFM 68K.

#### ***Macintosh 68K Code Resources***

Plug-ins packaged as code resources on 68K must include a 'Drop' resource, containing the main entry points. Optionally, some may include an A4-based 'Dhlp' code resource, for browser symbol generation in compilers, and for name unmangling in linkers.

Mac 68K code resource based plug-ins must use standard techniques for accessing globals. Global variable access from within 68K code resources is accomplished indirectly via the A4 register, rather than the usual A5 register. All plug-in entry points must set up the A4 register to point to global storage before using any global variables.

[Listing 4.1](#) illustrates how to set up Mac 68K A4-based global addressing.

---

#### **Listing 4.1    Macintosh 68K code resource A4 register setup**

---

```
#include <A4Stuff.h>           /* 68K code resources */

/* Example main entry point for a plug-in. */
CWPlugin_ENTRY (main) (CWPluginContext context)
{
```

---

```
CWResult result = cwNoErr;

/* set up global addressing (68K only, ignored for PowerPC) */
/* use EnterCodeResource() for Mac 68K code resource plug-ins */
*/
/* disable for other platforms */
EnterCodeResource();

/* ... */
/* get and handle IDE request here... */
/* ... */

/* report our error result to the IDE */
result = CWDonePluginRequest(context, result);

/* tear down global world (68K only, ignored for PowerPC) */
/* use ExitCodeResource() for Mac 68K code resource plug-ins */
/* disable for other platforms */
ExitCodeResource();
/* CAUTION: do not use globals after this point */

/* return CWDonePluginRequest's error result to IDE */
return (result);
}
```

---

### **Macintosh 68K CFM**

Mac 68K plug-ins implemented using CFM have the advantage that almost no variation in source is required to accommodate both 68K and PowerPC versions of a plug-in. CFM 68K based plug-ins export their entry points by name in the same way PowerPC plug-ins do. In contrast, 68K code resource-based plug-ins implement one code resource per entry point, and must provide information about their capabilities using resources, rather than informational entry points.

Plug-ins implemented using CFM 68K are not supported on PowerPC, so any plug-in that uses CFM 68K exclusively will run only on 68K machines. Plug-ins that include both 68K and PowerPC code fragments will run on both architectures. Plug-ins that use 68K code resources, with or without PowerPC code fragments, will run on both 68K and PowerPC.

### **Macintosh Code Fragment ('cfrg') Resources**

All CFM-based plug-ins should include a 'cfrg' resource that describes the PowerPC code fragment in the plug-in's data fork, and also the 68K code fragment, if the 68K code is also packaged as a CFM code fragment. The required 'cfrg' resources will be generated automatically by the Mac OS Merge linker when it combines the output of the 68K and PowerPC linkers to produce the fat binary.

The sample projects illustrate the construction of a fat plug-in from 68K and PowerPC code. You are encouraged to clone a sample project, rather than trying to construct a fat plug-in project manually.

## **Platform Differences in Plug-in API**

The CodeWarrior Plug-in API version 9 and above (introduced with CodeWarrior Pro 3) can be used on both Windows and Mac OS hosted platforms to create compiler and linker plug-ins. Core API services are uniform across host platforms with a few minor exceptions.

### **Resources on Mac OS**

In early versions of the plug-in API, Macintosh plug-ins used resources to provide capability information to the IDE. Starting with version 9 of the API, both Mac OS and Windows plug-ins may use a common set of informational entry points to provide the same information, enhancing plug-in portability.

Use of resources is still supported on Mac OS. However, it is recommended that all plug-ins use the informational entry points to simplify cross-platform development. Future extensions to the informational entry points may not be supported with corresponding resource changes.

---

**NOTE**

Plug-ins packaged as Mac OS 68K code resources require the use of informational resources, because the IDE cannot find entry points (other than `main`) in 68K code resources.

---

To maintain plug-ins which still use resources or to develop plug-ins that run on older versions of the IDE, see the resource format descriptions listed in [“Mac OS Plug-in Resource Formats” on page 641](#). For more details on the informational entry points, see [“Specifying Plug-in Capabilities” on page 65](#).

---

**NOTE** The GetDropInName and GetDisplayName entry points have been superceded by GetPluginInfo. GetDropInName and GetDisplayName continue to work, but new development should use GetPluginInfo.

---

## Platform Dependent API Differences

Some of the information required to operate plug-ins on both Windows and Mac OS hosted platforms is by necessity platform dependent. These differences are handled by conditional `typedefs` appearing in `CWPlugin.h`.

[Listing 4.2](#) illustrates all the basic types whose definition is platform dependent. Other types incorporating these types also vary by platform ([CWFileInfo](#), [CWMessageref](#), [CWAcessPathInfo](#), [CWDependencyInfo](#), [CWObjectData](#), [CWTargetInfo](#), and [CWProjectFileInfo](#)).

### **Listing 4.2 Platform specific types**

---

```
#if CWPLUGIN_API == CWPLUGIN_API_MACOS

#if CWPLUGIN_LONG_FILENAME_SUPPORT

#define CWPLUGIN_FILENAME_LEN 256
typedef struct CWFfileSpec
{
    FSRef         parentDirRef; /* parent directory */
    HFSUniStr255 filename;     /* unicode file name */
} CWFfileSpec;

typedef char CWFfileName[CWPLUGIN_FILENAME_LEN];

#else
#define CWPLUGIN_FILENAME_LEN 32
```

---

```
typedef FSSpec CWFfileSpec;
typedef char    CWFileName[CWPLUGIN_FILENAME_LEN];

#endif

typedef unsigned long CWFfileTime;
typedef OSerr          CWOSResult;

#if CWPLUGIN_API == CWPLUGIN_API_WIN32

typedef unsigned char Boolean;
typedef struct CWFfileSpec { char path[MAX_PATH]; } CWFfileSpec;

#if CWPLUGIN_LONG_FILENAME_SUPPORT

#define CWPLUGIN_FILENAME_LEN 256
typedef char CWFileName[CWPLUGIN_FILENAME_LEN];

#else

#define CWPLUGIN_FILENAME_LEN 65
typedef char CWFileName[CWPLUGIN_FILENAME_LEN];

#endif

typedef FILETIME CWFfileTime;
typedef DWORD    CWOSResult;

#elif CWPLUGIN_API == CWPLUGIN_API_UNIX

#define MAX_PATH           MAXPATHLEN
#define CWPLUGIN_FILENAME_LEN 65
#ifndef __MACTYPES__
    typedef unsigned char Boolean;
#endif

#ifndef FALSE
    #define FALSE 0
#endif

#ifndef TRUE
    #define TRUE 1

```

```
#endif

typedef struct CWFfileSpec { char path[MAX_PATH]; } CWFfileSpec;
typedef char CWFfileName[CWPLUGIN_FILENAME_LEN];
typedef time_t CWFfileTime;
typedef int CWOSResult;

#else

#error Unknown Plugin API!

#endif
```

---

If you need to store one of these types in a file, or otherwise transmit it between platforms, you will have to provide extra code to translate the information represented by these types from one platform to another. Usually, this is not necessary, since the IDE handles information stored in project and target files.

## Getting Runtime Information

The IDE provides two routines for obtaining information about the plug-in runtime environment: [CWGetIDEInfo](#) and [CWGetAPIVersion](#).

Most plug-ins will find that they can adequately specify their dependence upon IDE services by indicating the earliest and latest IDE versions with which they can operate properly using the `CWPlugin_GetDropInFlags` plug-in entry point (see [Specifying Plug-in Capabilities](#)).

Some plug-ins, however, may adapt to variations in the level of service provided by different versions of the IDE. To do this, they may need to know the specific version of the IDE they are running under.

The IDE provides a routine for checking the plug-in API version in effect at runtime. [CWGetAPIVersion](#) retrieves the version of the

API that the IDE is presenting to the plug-in. This version may not be the latest supported by the IDE, in either of the following cases:

1. The plug-in is running under a later version of the API than it was developed for.
2. The plug-in explicitly states its ability to run under older versions of the API (which will happen when the plug-in is used with an older IDE version).

Plug-ins may also use the [`CWGetIDEInfo`](#) routine to determine the version of the IDE, as well as the latest supported API version (without regard to the API version currently being presented to the plug-in).

Most plug-ins should not require these routines. You are encouraged to avoid writing version-specific plug-in code, where possible.

## Informational Entry Points

When the IDE loads a plug-in, it calls various plug-in entry points to determine more about the plug-in. The plug-in entry points pass information back to the IDE that describes the plug-in, the services it provides, its names, and its associated settings panels.

This section covers the following topics:

- [Specifying Plug-in Capabilities](#)
- [Specifying a Plug-in's Dropin and Display Names](#)
- [Specifying Associated Settings Panels](#)
- [Specifying Panel Family](#)

### Specifying Plug-in Capabilities

The IDE determines the kinds of operations performed by a plug-in by calling its `CWPlugin_GetDropInFlags` entry point. This routine is exported by the plug-in, and informs the IDE of the plug-in's capabilities.

The CWPlugin\_GetDropInFlags entry point is provided by all compiler, linker, version control, and settings panel plug-ins, and has the following declaration:

```
CWPlugin_ENTRY(CWPlugin_GetDropInFlags)
    (const DropInFlags** flags, long* flagsSize);
```

Although the declaration of this routine is identical for all plug-in types, the information it returns differs for settings panels. All plug-ins except settings panels return a pointer to a [DropInFlags](#) structure. Settings panel plug-ins return a pointer to a [PanelFlags](#) structure, cast to a [DropInFlags](#) pointer.

The initial members of both structures include information about the type of the plug-in, which allows the IDE to infer the layout of succeeding members. This facilitates returning different information from plug-ins using the uniform declaration of CWPlugin\_GetDropInFlags.

### **Specifying Compiler, Linker, and Version Control Plug-in Capabilities**

The CWPlugin\_GetDropInFlags entry point for compiler, linker and VCS plug-ins returns a pointer to a DropInFlags structure, which describes the capabilities of a plug-in. The DropInFlags structure is defined in [Listing 4.3](#):

#### **Listing 4.3 DropInFlags Structure**

---

```
typedef struct DropInFlags {
    short            rsrcversion;
    CWDataType      dropintype;
    unsigned short  earliestCompatibleAPIVersion;
    unsigned long   dropinflags;
    CWDataType      edit_language;
    unsigned short  newestAPIVersion;
} DropInFlags, **DropInFlagsHandle;
```

---

A [DropInFlags](#) struct specifies the following:

- The version of the structure
- The plug-in type

- The earliest and most recent API versions the plug-in works properly with
- Various bit flags indicating plug-in capabilities, the meanings of which differ according to the plug-in type
- For compilers, a four-character code indicating the language accepted by the compiler

The flags returned in `dropinflags` have different meanings for compilers, linkers, and VCS plug-ins. For more information about the flags for a specific plug-in type, see the following:

- For information about compiler flags, see “[Compiler Capability Flags](#)” on page 200
- For information about linker flags, see “[Linker Capability Flags](#)” on page 203
- For information about VCS flags, see “[VCS Plug-in Capability Flags](#)” on page 536

Plug-ins should use one of the following four-character codes to indicate plug-in type in the `dropintype` field (note the space in some of these in order to occupy the four characters):

Plug-in type	<code>dropintype</code> value
Compiler	'Comp '
Linker	'Link'
Version control system	'VCS '
COM plug-in	'COM '

Defines for these values appear in `CWPlugins.h`.

See [DropInFlags](#) for a complete description of `DropInFlags`. See [Listing 4.4](#) for an example of how to fill in and return this structure in the `CWPlugin_GetDropInFlags` routine.

#### **Listing 4.4 CWPlugin\_GetDropInFlags example**

---

```
CWPlugin_ENTRY (CWPlugin_GetDropInFlags)
    (const DropInFlags** flags, long* flagsSize)
{
    static const DropInFlags sFlags =

```

---

## Creating CodeWarrior IDE Plug-ins

### Specifying Plug-in Capabilities

---

```
{  
    kCurrentDropInFlagsVersion,      // rsrcversion  
    CWDROPINCOMPILERTYPE,          // dropintype  
    DROPINCOMPILERLINKERAPIVERSION_7,  
                                // earliestCompatibleAPIVersion  
    kCompMultiTargAware,           // dropinflags  
    Lang_C_CPP,                  // edit_language  
    DROPINCOMPILERLINKERAPIVERSION // newestAPIVersion  
};  
  
*flags = &sFlags;  
*flagsSize = sizeof(sFlags);  
  
return cwNoErr;  
}
```

---

Mac OS You can define a 'Flag' resource that maintains identical DropInFlags information. See "["Flag" Resource on page 645](#)" for more information.

**NOTE** Use of CWPlugin\_GetDropInFlags overrides the use of a 'Flag' resource.

---

### Specifying Preference Panel Capabilities

The CWPlugin\_GetDropInFlags entry point for preference panel plug-ins returns a pointer to a PanelFlags structure, which describes the capabilities of a plug-in. This structure is similar to but not the same as the DropInFlags structure returned by other plug-in types. The PanelFlags structure is defined in [Listing 4.5](#):

#### **Listing 4.5** PanelFlags Structure

---

```
typedef struct PanelFlags {  
    unsigned short    rsrcversion;  
    CWDATAType       dropintype;  
    unsigned short    earliestCompatibleAPIVersion;  
    unsigned long     dropinflags;  
    CWDATAType       panelfamily;  
    unsigned short    newestAPIVersion;  
    unsigned short    dataversion;
```

---

```
    unsigned short    panelscope;
} PanelFlags;
```

---

A `PanelFlags` structure specifies some of the same information as `DropInFlags`, including the API version range the plug-in is compatible with, and various flags. `dropintype` indicates the plug-in type (settings panels use 'PanL'), and `dropinflags` indicates capabilities and properties of the plug-in. `PanelFlags` also indicates the version of a plug-in's data, the plug-in family (what panel group it appears with), and the scope of its plug-in (in what contexts it appears in the target settings dialog).

See [“PanelFlags Structure” on page 340](#) for a complete description of `PanelFlags`. See [Listing 4.6](#) for an example of how to fill in and return this structure in a preference plug-in’s `CWPlugin_GetDropInFlags` routine.

#### **Listing 4.6 Preference Panel CWPlugin\_GetDropInFlags example**

---

```
CWPlugin_ENTRY(CWPlugin_GetDropInFlags)
    (const DropInFlags** flags, long* flagsSize)
{
    static const PanelFlags sFlags = {
        3,                                // rsrcversion
        CWDROPINPREFSTYPE,                // dropintype
        DROPINPANELAPIVERSION_7,          // earliestCompatibleAPIVersion
        usesStrictAPI | supportsByteSwapping | supportsTextSettings, // dropinflags
        kSamplePanelFamily,               // panelfamily
        DROPINPANELAPIVERSION,            // newestAPIVersion
        PSAMPLEPANELVERSION,              // dataversion
        panelScopeTarget                 // panelscope
    };

    *flags = (DropInFlags*) &sFlags;
    *flagsSize = sizeof(sFlags);

    return cwNoErr;
}
```

---

**NOTE** The pointer to the `PanelFlags` struct is cast to type `DropInFlags` prior to return. This allows `CWPlugin_GetDropInFlags` to have the same definition for all plug-in types.

---

Mac OS You can define a 'Flag' resource that maintains identical `DropInFlags` information. See ['Flag' Resource](#) for more information.

**NOTE** Use of `CWPlugin_GetDropInFlags` overrides the use of a 'Flag' resource.

---

## Specifying a Plug-in's Dropin and Display Names

The IDE uses two different names to identify plug-ins. One is the 'dropin' name; this name is used internally by the IDE to keep track of plug-ins. The other name is the 'display' name; this name is used when the IDE must display the name of a plug-in to the user (this happens, for example, when selecting compilers and linkers in **Target Settings** under the **Edit** menu).

The dropin and display names will usually differ if you produce localized versions of your plug-in. All localized versions of your plug-in should share a common dropin name, but have localized display names (as well as any additional text displayed to the IDE user). This allows users of different localized plug-in versions to share projects without difficulties.

### Specifying Dropin Name

Each of these names is returned by a different plug-in entry point. `CWPlugin_GetDropInName` returns the dropin name, and `CWPlugin_GetDisplayName` returns the display name. Both routines simply return a pointer to a constant C string containing the name. The pointers returned must remain valid for the duration of the plug-in's execution (the entire time it is loaded).

A `CWPlugin_GetDropInName` entry point is declared as follows:

```
CWPlugin_ENTRY (CWPlugin_GetDropInName)
    (const char** dropinName)
```

[Listing 4.7](#) shows an example of how to implement the dropin name entry point.

#### **Listing 4.7 CWPlugin\_DropInName example**

---

```
CWPlugin_ENTRY(CWPlugin_GetDropInName)(const char** dropinName)
{
    static const char* sDropInName = "Sample Compiler";
    *dropinName = sDropInName;
    return cwNoErr;
}
```

---

#### **Specifying Display Name**

A CWPlugin\_GetDisplayName entry point is declared as follows:

```
CWPlugin_ENTRY (CWPlugin_GetDisplayName)
                (const char** dropinName)
```

[Listing 4.8](#) shows an example of how to implement the display name entry point.

#### **Listing 4.8 CWPlugin\_GetDisplayName example**

---

```
CWPlugin_ENTRY(CWPlugin_GetDisplayName)(const char** displayName)
{
    static const char* sDisplayName= "Sample Compiler Display
Name";
    *displayName = sDisplayName;
    return cwNoErr;
}
```

---

**Mac OS** To specify the character string that the IDE uses to display a plug-in's name on-screen, use a 'STR' resource. See "['STR' Resource](#)" for more information.

**NOTE** Use of CWPlugin\_GetDisplayName overrides the use of a 'STR' resource.

---

## Specifying Associated Settings Panels

To inform the IDE which preference panels the plug-in requires, use:

```
CWPlugin_ENTRY(CWPlugin_GetPanelList)
    (const CWPanellist** panelList)
```

The CWPlugin\_GetPanelList entry point returns a list of settings panels that the IDE should make available when a plug-in is in use by a project's current target. Settings panels are identified by name, and the list returned allows a plug-in to specify as many panel dependencies as necessary.

### **Listing 4.9 CWPlugin\_GetPanelList example**

---

```
CWPlugin_ENTRY(CWPlugin_GetPanelList)
    (const CWPanellist** panelList)
{
    static const char* sPanelName = "Sample Panel";
    static CWPanellist sPanelList = {kCurrentCWPanellistVersion,
                                    1, &sPanelName};

    *panelList = &sPanelList;
    return cwNoErr;
}
```

---

Only compilers, linkers, pre-linkers, post-linkers, and version control plug-ins implement this entry point, since settings panels do not specify additional settings panel associations.

**Mac OS** You can use a 'STR#' resource to list the settings panels required by a plug-in. See "["STR#" Resource](#)" on page 652 for more information.

---

**NOTE** Use of CWPlugin\_GetPanelList overrides the use of a 'STR#' resource.

---

## Specifying Panel Family

The IDE provides a mechanism for grouping preference panels into “families” specifying related target settings. The IDE groups all plug-in panels having the same family code, as returned in the panel’s [PanelFlags Structure](#), and displays them under one heading in the **Target Settings** dialog.

The IDE obtains the names of the headings of all panel families by calling each plug-in that provides a family name entry point. This entry point is declared as follows:

---

### **Listing 4.10    Plug-in family name entry point**

---

```
CWPlugin_ENTRY (CWPlugin_GetFamilyList)
    (const CWFamillyList** familyList);
```

---

This entry point returns both a name and a family type, using the following structures:

---

### **Listing 4.11    Family name entry point data structures**

---

```
typedef struct CWFamilly {
    CWDataType      type;
    const char*     name;
} CWFamilly;

typedef struct CWFamillyList {
    short           version;
    short           count;
    CWFamilly*     families;
} CWFamillyList;
```

---

The CWFamillyList structure returns a list of CWFamilly structures, each of which specifies a family type and family name. The family type is a four-character code, and the name is a C string.

---

**NOTE** Family type codes consisting of all lower-case letters are reserved for use by Metrowerks. To avoid collisions with the family codes specified by other plug-in vendors, use distinctive type codes.

---

The IDE uses the family type to match family names with preference panels. When multiple preference panels specify the same family code as a family name entry point, the IDE displays the preference panels in a group labeled using the family name returned by the family name entry point.

Family name entry points may be implemented by plug-ins of any type. Typically, linkers are used to implement the family name entry point, since they determine which plug-ins are enabled, are a given suite of plug-ins usually contains one (and only one) linker.

In some cases, however, this may not be possible. For example, if you implement a compiler that generates code for an existing linker, you will need to implement the family entry point in some other plug-in, such as the compiler.

The IDE assumes that each family name entry point will return a unique family name. The IDE also assumes that for any family membership specified by a preference panel, a family name entry point will be found that provides the family name. Otherwise, the IDE will use a family name of “Other.”

In many cases, you may wish to group a settings panel in one of the standard Metrowerks settings panel families. Use one of the type codes listed in [Table 4.1](#).

**Table 4.1 Standard settings panel family type codes**

To group a panel in this family:	Use this code:
Target	‘proj’
Editor	‘edit’
Language Settings	‘fend’
Code Generation	‘bend’
Linker	‘link’
Version Control	‘vcs’
Debugger	‘dbug’
Browser	‘brow’

To group a panel in this family:	Use this code:
RAD Tools	'radt'
Miscellaneous	'****'

## Responding to IDE Requests

Most of the significant functionality implemented by a plug-in is provided to the IDE in response to requests from the IDE, delivered to a plug-in's main entry point (which is usually a C-style `main()` function). This section describes the IDE requests sent to all plug-in types and how to respond to them.

This section covers the following topics:

- [Basic Request Handling](#)
- [Returning Error Results to the IDE](#)
- [Common Requests](#)
- [Reentrancy Considerations](#)

### Basic Request Handling

IDE requests are made to a plug-in's main routine which should be declared as in listing [Listing 4.12](#). The main entry point for all plug-ins is passed a single `context` parameter, and returns a short error code. The entry point obeys `__stdcall` calling conventions on Windows and Pascal calling conventions on Mac.

---

#### **Listing 4.12    Plug-in main () entry point declaration**

---

```
CWPlugin_ENTRY (main) (CWPluginContext context)
```

---

The plug-in main entry point should do the following:

1. determine the request made by the IDE
2. take appropriate action
3. return an error result

To determine the request being made by the IDE, a plug-in calls [CWGetPluginRequest](#), which returns a code indicating the request type. A plug-in usually then dispatches execution to a subroutine appropriate for handling a particular request. [Listing 4.13](#) illustrates this process.

**Listing 4.13 Handling requests from the IDE**

```
/* Main entry point for a plug-in. */
/* This main() function is the plug-in's request handler. */

CWPlugin_ENTRY (main) (CWPluginContext context)
{
    CWResult    result;
    long        request;

    /* get the current pending request from the IDE */
    result = CWGetPluginRequest(context, &request);
    /* if no error, dispatch request */
    if (result == cwNoErr)
    {
        /* dispatch on request */
        switch (request)
        {
            case reqInitialize:
                /* plug-in has just been loaded into memory */
                MyInitPlugin();
                break;

            case reqTerminate:
                /* plug-in is about to be unloaded from memory */
                MyTerminatePlugin();
                break;

            /* ... */
            /* handle other requests here */
            /* ... */

            default:
                /* let IDE know we don't support this request */
                result = cwErrRequestFailed;
                break;
        }
    }
}
```

```
        }
    }
else
    result = cwErrRequestFailed; /* couldn't get request */

/* report our error result to the IDE */
result = CWDonePluginRequest(context, result);

/* return CWDonePluginRequest's error result to IDE */
return (result);
}
```

---

The specific requests made to a plug-in vary by plug-in type. Also, some requests are not made unless a plug-in indicates that it wants them. To indicate its interest in optional requests, a plug-in sets the appropriate flags in its `CWPlugin_GetDropInFlags` entry point (see [“Specifying Plug-in Capabilities” on page 65](#)).

### Main Entry Point `context` Parameter

The `context` parameter passed to the plug-in maintains state private to the IDE required to interact with a plug-in. The `context` parameter must be passed back to the IDE in all calls made by the plug-in. Plug-ins should never attempt to inspect or modify the state maintained by this parameter.

To avoid problems with reentrancy in current and future versions of the IDE, plug-ins should always pass `context` as a parameter to internal routines, rather than caching it globally. This is because `context` may vary from one IDE request to the next.

## Returning Error Results to the IDE

After servicing a request, and immediately before returning execution to the IDE, a plug-in calls `CWDonePluginRequest` with an error code of type `CWResult` that describes the plug-in’s success in handling a request. The plug-in should then pass the value returned by `CWDonePluginRequest` back to the IDE as its `main()` function result. [Listing 4.14](#) illustrates this process.

**Listing 4.14    Returning an error result to the IDE**

```
CWPlugin_ENTRY (main) (CWPluginContext context)
{
    CWResult    result = cwNoErr;

    /* ... */
    /* get and handle IDE request here... */
    /* set result to the proper error value in the process */
    /* ... */

    /* report our error result to the IDE */
    result = CWDonePluginRequest(context, result);

    /* return CWDonePluginRequest's error result to IDE */
    return (result);
}
```

---

**NOTE**    Plug-ins should return `cwErrRequestFailed` in response to requests from the IDE that they do not understand.

---

The error codes for compiler and linker plug-ins that the IDE recognizes are listed in [“Result Codes for Compiler and Linker Plug-ins”](#) and [“Result Codes for Plug-ins”](#).

## Common Requests

Some plug-in requests are sent by the IDE to all plug-ins, regardless of plug-in type. This section briefly describes them.

### **reqInitialize Request**

The IDE sends this request immediately after a plug-in is loaded for use, and before any other requests are issued. This gives a plug-in a chance to set up global state, and to possibly acquire any necessary global resources. Such resources must be acquired using OS-specific calls, since most IDE services, including all memory allocation services, are *not* available during this request. Many plug-ins will have little or nothing to do in response to this request.

**NOTE** While handling a [reqInitialize](#) request, a plug-in may only call [CWGetPluginRequest](#) and [CWDonePluginRequest](#).

---

Note that If a plug-in wishes to allocate data that persists globally between instances of loading and unloading by the IDE, the plug-in must use target data storage facilities when responding to a request (see [“Storing and Retrieving Plug-in Data”](#)).

**NOTE** Target data storage is *not* available to settings panel plug-ins.

---

Global variable initializations made during the [reqInitialize](#) request will persist only until a plug-in is unloaded (that is, until a [reqTerminate](#) request is made). When a plug-in is reloaded, its global state is reset to its initial state, not preserved.

### **reqTerminate Request**

The IDE sends this request prior to unloading a plug-in. Usually, the plug-in will undo whatever it did in response to the reqInitialize request, such as releasing any acquired resources. Since plug-ins cannot allocate memory or make other calls in response to this request or the [reqInitialize](#) request, it follows that the plug-in may only free resources acquired using OS-specific calls.

**NOTE** While handling a [reqTerminate](#) request, a plug-in may only call [CWGetPluginRequest](#) and [CWDonePluginRequest](#).

---

### **Additional Requests**

Each type of plug-in responds to additional requests besides the common requests described here. See the following sections for more information:

---

<b>For information about:</b>	<b>See:</b>
compiler and linker requests	<a href="#">“Handling Compiler and Linker Requests” on page 211</a>

---

For information about:	See:
version control system requests	<a href="#">“VCS Requests” on page 556</a>
preference panel requests	<a href="#">“Settings Panel Requests” on page 345</a>

## Reentrancy Considerations

In current versions of the CodeWarrior IDE, it is possible for the IDE to make multiple simultaneous calls to a plug-in. This could in theory happen for a variety of reasons (and may in the future). XXX

In practice, the IDE currently only makes multiple simultaneous plug-in calls in one case: when a linker plug-in is servicing a `reqTargetInfo` request. If a linker supports returning target information during a link request, it should set the `linkerGetTargetInfoThreadSafe` flag in its [DropInFlags](#).

Supporting concurrent requests, while not required, will usually result in better plug-in behavior, and will probably make it easier to adapt to future changes in the CodeWarrior Plug-in API. Doing so is usually relatively easy for simple requests, such as `reqTargetInfo`.

To support concurrent plug-in requests reliably, avoid using global plug-in state, whether maintained in actual global variables, or in operating system state. Instead, store state locally on the stack. When global data access is necessary, use synchronization primitives to ensure data coherency and exclusive access by a single thread (assume each call to the plug-in is made by a different IDE thread), and be careful to avoid deadlock situations.

## Managing Memory

The CodeWarrior IDE manages memory for itself and plug-ins. It provides routines for managing memory which all plug-ins should use, rather than operating system memory management services. Most plug-in writers should read this section carefully; it contains important information about managing memory effectively in the CodeWarrior plug-in environment.

This section covers the following topics:

- [Kinds of Memory](#)
- [Pointers](#)
- [Allocating and Disposing Pointers](#)
- [Handles](#)
- [Allocating and Disposing Handles](#)
- [Accessing Handle Memory](#)
- [Resizing Handles](#)
- [Pointers versus Handles](#)

## **Kinds of Memory**

The CodeWarrior API supports two different kinds of memory for use by plug-ins, for their own purposes, and for communicating with the IDE. These two types of memory are known as pointer memory and handle memory, or when referring to blocks of such memory, as pointers and handles, respectively.

---

**NOTE** Plug-ins should use CodeWarrior for all memory management, rather than using native operating system or runtime library memory management routines such as `malloc()`.

---

## **Pointers**

Pointers, which refer to pointer memory, or pointer blocks, are like any other pointer in computer science. They refer to some amount of contiguous memory which resides at a fixed address for the duration of the memory block's lifetime.

To refer to pointer memory obtained using the CodeWarrior API, simply dereference it as you would any other pointer. There are no complexities involved in using CodeWarrior pointers.

## **Allocating and Disposing Pointers**

The API provides only two routines for dealing with pointers: [CWAllocateMemory](#) and [CWFreeMemory](#). [CWAllocateMemory](#)

attempts to allocate a pointer block of the size requested, and returns an error code indicating success or failure. [CWFreeMemory](#) frees a valid pointer allocated using [CWAllocateMemory](#). Do not call [CWFreeMemory](#) twice for the same pointer.

By default, all memory allocated using [CWAllocateMemory](#) is disposed automatically by the API on return from any plug-in entry point. In some cases, this may not be desirable. For example, a plug-in may compute a lookup table used for the duration of its operation. For such purposes, it may wish to retain blocks of memory across calls to the plug-in by the IDE.

[CWAllocateMemory](#) provides a parameter (`isPermanent`) that supports allocating memory permanently. Memory allocated with `isPermanent` set to true will not be automatically deallocated by the IDE. Blocks allocated with `isPermanent` set to true are retained by the IDE until they are explicitly deallocated by the plug-in, even if the plug-in is unloaded from memory.

plug-ins must be careful to dispose of this memory using [CWFreeMemory](#) to avoid causing memory leaks. In addition, the same value must be specified for `isPermanent` when deallocating a pointer as was specified when allocating it. Usually, the cleanest way to manage memory allocated using `isPermanent` is to allocate it when the IDE makes a [reqTargetLoaded](#) call to the plug-in, and to dispose it during a [reqTargetUnloaded](#) call.

## Handles

Handles are blocks of memory of type [CWMemHandle](#) which, for historical reasons relating to managing memory effectively, are referred to indirectly. Handles are opaque data structures which cannot be dereferenced directly to access the memory they represent (unlike pointers). The IDE provides routines that support accessing handle memory.

---

**NOTE** Plug-ins should make no assumptions about the contents or structure of a handle block. All operations on handles should be performed using the CodeWarrior Plug-in API.

---

Most IDE routines that require dynamically allocated memory use handles. Routines like [CWStoreObjectData](#) and [CWGetNamedPreferences](#) require blocks of type [CWMemHandle](#).

## Accessing Handle Memory

Unlike pointers, handles cannot be dereferenced directly. Instead, to use a handle, a plug-in must first call [CWLockMemHandle](#) to obtain a pointer to the memory in the handle.

Calling [CWLockMemHandle](#) returns a pointer guaranteed to be valid until [CWUnlockMemHandle](#) is called. This pointer may be dereferenced like any other pointer to access a handle's contents. Calling [CWUnlockMemHandle](#) on a handle invalidates all existing pointers to the handle memory.

- 
- NOTE** Calls to [CWLockMemHandle](#) and [CWUnlockMemHandle](#) are counted. Therefore, pointers to handles may not actually become invalid upon calling [CWUnlockMemHandle](#); this depends upon whether a handle's lock counted has decreased to zero. But proper, safe practice is to always lock a handle before using its contents, and to assume that any pointers to it become invalid once the handle is unlocked.
- 

Because the contents of unlocked handle blocks may be relocated (moved to a different address), it is very important to obey careful discipline when accessing handles. If, for example, a pointer to a handle's contents is obtained by locking the handle, and later the handle is unlocked, but the pointer to its contents is retained and later written to, serious corruption of memory may occur if the handle has been moved (since the pointer no longer points to the block contents).

Bugs caused by using such so-called “stale” pointers can be very hard to find. The best way to avoid them is to adopt rigorous discipline when using handles: declare and obtain pointers to handles locally, and avoid passing them around or caching them. Lock all blocks before using them, and unlock them immediately afterwards.

**NOTE** Do not lock blocks longer than necessary, as this can impair effective memory management by the IDE.

---

[Listing 4.15](#) shows an example of dereferencing a [CWMemHandle](#).

### **Listing 4.15 Dereferencing a CWMemHandle**

---

```
typedef struct myStruct
{
    int myInt;
    /* ... */
} myStruct, * myStructPtr;

err = CWLockMemHandle(context, myHandle, true, &myPtr);
if (err == cwNoErr)
{
    /* access the handle contents using myPtr here */
    (myStructPtr)myPtr->myInt = 42;
}
/* when done, unlock the handle */
err = CWUnlockMemHandle(context, myHandle);
```

---

## **Allocating and Disposing Handles**

To allocate a handle, use [CWAllocMemHandle](#). Note that all handles allocated by a plug-in are automatically deallocated when the plug-in completes the currently pending request (that is, when it returns from the `main()` entry point). Before using a newly-allocated handle, be sure to verify that the handle was successfully allocated by examining the error code from [CWAllocMemHandle](#).

To dispose a handle, use [CWFreeMemHandle](#). Do not call [CWFreeMemHandle](#) more than once for the same handle. Note that even though handles are automatically deallocated on return to the IDE, you may still want or need to dispose handles prior to return (especially when dealing with large allocations).

## Resizing Handles

Unlike pointer blocks, handle blocks can be resized using [CWResizeMemHandle](#). In addition, you may obtain the current size of a handle by calling [CWGetMemHandleSize](#).

A handle is commonly used to accomodate dynamically-sized data, by obtaining its size using [CWGetMemHandleSize](#) and then changing its size by a fixed amount or a percentage (for better performance when memory isn't at a premium) using [CWResizeMemHandle](#).

Since resizing a handle can fail, just like allocating one, it is important to check for errors returned by [CWGetMemHandleSize](#). Also, resizing a handle is much more likely to succeed when the handle is unlocked.

## Pointers versus Handles

When deciding whether to use pointers or handles, consider the following points:

- pointers cannot be used with most of the plug-in API
- handles are usually better for resizable blocks; you cannot determine the size of a pointer block, or resize it easily
- on some platforms, handles may reduce memory fragmentation
- pointers are simpler and much less error prone to use, when they are an option
- although usually insignificant, pointers may consume slightly less memory, and have less performance overhead

## Managing Files

The IDE provides routines for finding, loading, and modifying files in a project, and getting information about files, targets, and projects.

This section covers the following topics:

- [Project Structure](#)
- [Specifying Project Components](#)

- [File Specifications](#)
- [Obtaining Information About Files](#)
- [Getting a File](#)
- [Getting Files Not in the Project](#)
- [Modifying Files in the Project](#)
- [Getting Information About Projects](#)
- [Getting Information About Targets](#)
- [Getting Information About Overlays and Segments](#)
- [Getting Information About Project Paths](#)

## Project Structure

While communicating with plug-ins, the IDE always maintains a current project. This project is the object of most user actions, and is the sole project visible to a plug-in. Most file services provided by the API apply to the current project.

Projects may contain numerous components:

- *Files*, which specify the source code for a project.
- *Groups*, which organize files into logically related groups for human benefit. Each group may contain multiple files.
- *Targets*, which specify a final platform-specific output, produced by a linker, and which source files contribute to it
- *Subprojects*, which have all the properties of projects, and which are supported by the IDE's "make" intelligence
- *Overlays*, which specify code to be loaded into memory at a specific address, used for embedded targets only. Overlays contain files, each of which contributes to a binary image loaded into memory at the address of the overlay's parent group.
- *Overlay groups*, which specify a collection of overlays and their load address, used for embedded targets only.
- *Segments*, which specify the code collected into one 32K compiled 'CODE' resource, used for Macintosh 68K targets only.

The IDE provides facilities for enumerating files, overlays, overlay groups, and segments (but not targets, groups, or subprojects). The

IDE also provides routines for obtaining or modifying properties of all of the above, except groups and subprojects.

## Specifying Project Components

When calling an IDE service that returns or modifies information about a project component, the plug-in must specify an index for the component. This index is always zero-based (that is, the first component has index zero), and indexes only those components belonging to the current target. Thus, the maximum file, overlay, or segment index may vary by project target.

## File Specifications

When specifying the locations of files, the IDE uses a structure of type [CWFileSpec](#). The definition of this structure varies by platform, but always holds sufficient information to consistently locate a file on the host operating system.

- |         |  |
|---------|--|
| Windows | On Windows, this is a full file path.  |
| Mac OS  | On Mac OS, this is a <code>FileSpec</code> record. Because of this, <code>FileSpecs</code> that refer to either folders or files within folders remain valid even if the parent folder is renamed or moved to a different location on the same volume. |

## Obtaining Information About Files

To obtain information about files in the current project target, plug-ins may call [CWGetFileInfo](#), which returns extensive information about a file in a [CWPprojectFileInfo](#) structure.

Information returned includes the location of the file on disk, the file type, modification dates for the file and its corresponding object code, numerous flags about the file and how it is used in the current target, and the name of the plug-in that handles this file.

A [CWPprojectFileInfo](#) structure also contains some specialized information used by uncommon plug-in types, including segment information, used by Mac 68K target linkers, and a unit dependency checksum, used by Pascal compilers.

## Getting a File

To retrieve the contents of a file in the project, a plug-in calls [CWGetFileInfo](#) to get the file specification of the file, then calls [CWGetFileText](#) to load the file's contents into memory. Use the `filedatatype` argument returned by [CWGetFileText](#) to determine what kind of data is in the file: text ([cwFileTypeText](#)), precompiled data ([cwFileTypePrecompiledHeader](#)), or some other kind of data ([cwFileTypeUnknown](#)).

After processing the contents of a file, a plug-in should call [CWReleaseFileText](#) to deallocate the memory occupied by the file's contents.

## Getting Files Not in the Project

To retrieve a file that isn't necessarily in the project, but is available within the active target's access paths, a plug-in calls [CWFindAndLoadFile](#). As with [CWGetFileText](#), a plug-in may examine the `filedatatype` field of the [CWFileInfo](#) data structure returned by [CWFindAndLoadFile](#) to determine the kind of data in the file.

After processing the contents of the file, a plug-in should call [CWReleaseFileText](#) to deallocate the memory occupied by the file's contents.

## Modifying Files in the Project

A plug-in tells the IDE that a file in the project has changed by calling [CWSetModDate](#). If the plug-in has created the file rather than modifying a file already in the project, it must set the `isGenerated` argument to true.

To add a new file to a project, use [CWAddProjectEntry](#). If the file is generated from another project file, set the `isGenerated` parameter to true. Otherwise, if the file was created by some other source, and already exists on disk, specify false for `isGenerated`.

If a plug-in needs access to a project file that may currently be open for display to the user, the plug-in should call [CWPreFileAction](#) before opening the file, and [CWPostFileAction](#) after using and

closing the file. Calling [CWPreFileAction](#) tells the IDE to close the specified file. Calling [CWPostFileAction](#) afterwards informs the IDE that it may reopen the specified file.

---

**NOTE** [CWPreFileAction](#) and [CWPostFileAction](#) currently only work for VCS plug-ins; this may change in the future.

---

## Getting Information About Projects

To obtain the [CWFileSpec](#) of the current project file, plug-ins call [CWGetProjectFile](#). This may be useful in tracking the location or identity of a project file.

Plug-ins should avoid modifying or inspecting project files directly, and instead use plug-in API routines whenever possible.

To determine the number of files in the current project, a plug-in calls [CWGetProjectFileCount](#). Valid [CWGetFileInfo](#) file indexes range from zero to one less than the count returned.

## Getting Information About Targets

To get the name of a target, plug-ins call [CWGetTargetName](#). To determine the location specified by the user for target output in **Target Settings**, plug-ins call [CWGetOutputFileDirectory](#).

Plug-ins may also obtain the directory where the IDE stores target data files by calling [CWGetTargetDataDirectory](#). This is useful when adapting command line tools, such as compilers, that store object code in files external to the project. In these cases, the plug-in can obtain the proper location for temporary target files from the IDE, and pass this location to the command line tool as a command line parameter. The IDE deletes temporary files stored in a target directory when the user selects **Project > Remove Object Code**.

## Getting Information About Overlays and Segments

Plug-ins that support embedded platforms may need to obtain information about the overlays in a project. To get the number of

overlay groups in the current target, plug-ins call [CWGetOverlay1GroupsCount](#).

To get information about a specific overlay group, call [CWGetOverlay1GroupInfo](#) with an index ranging from 0 to one less than the count of overlay groups. [CWGetOverlay1GroupInfo](#) returns the number of overlays in a group.

To get information about one overlay, call [CWGetOverlay1Info](#) with an index ranging from 0 to one less than the number of overlays. [CWGetOverlay1Info](#) returns the number of files in an overlay.

To obtain information about a single file in an overlay, plug-ins call [CWGetOverlay1FileInfo](#) with an index from 0 to one less than the number of files in an overlay. The result indicates the file index of the file (in target link order).

Plug-ins that support Mac OS 68K targets call [CWGetSegmentInfo](#) to obtain information about segments. The API currently provides no way to determine the number of segments in a project; plug-ins can simply call [CWGetSegmentInfo](#) with increasing segment indexes until it returns an error of cwErrUnknownSegment.

## Getting Information About Project Paths

To determine the number of access paths in a project, plug-ins call [CWGetAccessPathListInfo](#). To enumerate all the user or system paths in a project, plug-ins call [CWGetAccessPathInfo](#), specifying a path index ranging from 0 to one less than the number of paths of the corresponding type returned by [CWGetAccessPathListInfo](#).

To enumerate the subdirectories of a particular path, plug-ins call [CWGetAccessPathSubdirectory](#). Subdirectories of a folder can only be enumerated when recursive folder searching is enabled for the folder in the **Access Paths** panel.

Plug-ins can convert relative paths from [CWRelativePath](#) structures to file specifications using [CWResolveRelativePath](#). Relative paths are commonly used in preference panels to store the locations of project paths. Relative paths are often more portable

than absolute paths when moving projects between CodeWarrior installations.

## Managing Plug-in Data

Plug-ins may need to load and store additional data associated persistently with a project target. This section describes the IDE services for managing plug-in data.

This section covers the following topics:

- [Storing and Retrieving Plug-in Data](#)
- [Storing and Retrieving Preference Data](#)
- [Storing Data Externally](#)

### Storing and Retrieving Plug-in Data

Plug-ins may need to store various kinds of information with a project file or target. For example, a compiler might generate a table of entry points for compiled object code, and store each table with its corresponding file. Or, it might store a global table of symbolic information, mapping routine names to the files that define them.

The IDE provides services for storing information with individual project files, and with the project as a whole. The API refers to such data as “plug-in data.”

#### Storing Plug-in Data

Plug-ins store data persistently with files and targets using [CWStorePluginData](#). [CWStorePluginData](#) stores data on disk which persists until the user chooses **Project > Remove Object Code**.

The data stored by [CWStorePluginData](#) is maintained on either a per-file or per-target basis. To store data with a file, pass the file’s index as the `whichfile` parameter. To store data with the current target, instead pass [kTargetGlobalPluginData](#) for the `whichfile` parameter.

Data stored with a file or target may have any format. Plug-ins determine the format of stored data, not the IDE. If your plug-in

undergoes frequent revision, you may wish to include a version number in the stored data, to facilitate use by multiple plug-in versions.

Data stored by [CWStorePluginData](#) is identified by a four-character code of type [CWDataType](#). If necessary, plug-ins may store multiple blocks of data with a single file or target, by using different identifying codes and multiple calls to [CWStorePluginData](#).

- 
- NOTE** Neither the IDE nor Metrowerks arbitrates the assignment of data type codes. To help avoid conflicts with the data stored by other plug-ins, choose a distinctive data type code, and test your plug-in with other commonly used plug-ins. Also note that all stored data is “public,” in the sense that it can be accessed by other plug-ins if they request the same data type.
- 

### Retrieving Plug-in Data

Data stored using [CWStorePluginData](#) can be retrieved using [CWGetPluginData](#). The data is returned in a [CWMemHandle](#), which should be locked before use with [CWLockMemHandle](#), and disposed using [CWFreeMemHandle](#) when no longer needed by a plug-in.

## Storing and Retrieving Preference Data

Most plug-ins require associated preference panels and preference data (also known as settings data). This data is used to configure plug-in operation, and is stored on a per-target basis.

Preference data is constructed by a preference panel plug-in and passed back to the IDE via its main entry point in response to a [reqGetData](#) request from the IDE. plug-in data is stored in the project by the IDE and can be loaded and examined later by a another plug-in, such as a compiler, linker, or version control plug-in.

To load preference data, a plug-in calls [CWGetNamedPreferences](#). Unlike generic plug-in data, preference data is stored by name. Usually, a plug-in should use a

descriptive name for its data that includes part or all of its internal plug-in name. This helps avoid conflicts among plug-ins.

[Listing 4.16](#) shows an example of retrieving settings panel data.

#### **Listing 4.16    Retrieving settings panel data**

---

```
CWResult MyGetPrefs(CWPluginContext context, PrefStruct
*prefData)
{
    CWMemHandle          prefsHand;
    PrefStruct           *prefsPtr;
    CWResult             err;

    /* load the relevant prefs */
    err = CWGetNamedPreferences(context, kPanelName, &prefsHand);
    if (err != cwNoErr)
        return (err);

    err = CWLockMemHandle(
        context, prefsHand, false, (void**)&prefsPtr);
    if (err != cwNoErr)
        return (err);

    *prefData = *prefsPtr;

    err = CWUnlockMemHandle(context, prefsHand);
    if (err != cwNoErr)
        return (err);

    return err;
}
```

---

Note that the IDE does not interpret settings panel data, nor does it provide a standard format for such data. A plug-in must know how to interpret the data it obtains using [CWGetNamedPreferences](#). To facilitate sharing data between preference plug-ins and other plug-ins that use the preference data, it is common to declare preference data in a header file shared between plug-ins that use it.

As with plug-in data returned by [CWGetPluginData](#), the preference data loaded by [CWGetNamedPreferences](#) is returned

to the plug-in in a [CWMemHandle](#). This handle should be disposed using [CWFreeMemHandle](#) when no longer required by the plug-in, and should be locked prior to use by calling [CWLockMemHandle](#).

Plug-ins should be careful to avoid loading and caching preference data. Preference data can be changed by the user at almost any time, while the plug-in is loaded (it will not, however, change during a **Bring Up To Date** or **Make** operation). To be safe, plug-ins should load their preference data at the start of every request during which it is required by a plug-in.

---

**NOTE** The CWPlugins.h API header defines a `reqPrefsChange` request which appears to apply to all plug-in types. Currently, however, the IDE only sends `reqPrefsChange` requests to VCS plug-ins.

---

## Storing Data Externally

In some cases, especially when adapting command line tools to the IDE environment, plug-ins may find it necessary to store data with a project target, but externally to the project file. In such cases, a plug-in may call [CWGetTargetDataDirectory](#) to obtain the directory in which the IDE stores per-target data. The plug-in can then store data files in this directory, or specify this folder as an output destination in the command line passed to a command line tool.

---

**NOTE** Files stored in this directory will be deleted when the user selects **Project > Remove Object Code**. In general, plug-ins should call [CWStorePluginData](#) rather than creating additional files in the target data directory, when possible.

---

## Handling User Interaction

The IDE provides routines to allow plug-ins to interact with the user. These topics describe how a plug-in communicates with the user:

- [Showing an Editor Window](#)

- [Reporting Progress](#)
- [Reporting Errors and Warnings from Compilers and Linkers](#)
- [Reporting Other User Messages](#)
- [Displaying Plug-in Dialogs](#)
- [Checking for User Breaks](#)
- [Obsolete User Interface Routines](#)

## **Showing an Editor Window**

To present text to the user in an IDE document window, a plug-in calls [CWCreateNewTextDocument](#). For example, a compiler or linker plug-in uses this routine to present a disassembly of object data or preprocessed source code.

### **Reporting Progress**

The IDE has routines to present the user with status and progress information. A plug-in calls [CWShowStatus](#) to tell the user about the plug-in's current action, such as the name of the file it is currently processing or a short description of the step it is executing among a series of steps.

A compiler plug-in calls [CWDisplayLines](#) to display the number of source code lines it has processed. [CWDisplayLines](#) should be called at regular intervals. Calling [CWDisplayLines](#) too often will hinder the plug-in's performance, while calling it too seldom will give the user poor feedback on the plug-in's state.

### **Reporting Errors and Warnings from Compilers and Linkers**

To present an error or warning that occurred while processing source code or object data, a compiler or linker plug-in calls [CWReportMessage](#). The messages are displayed in the IDE's message window, rather than in a dialog. [CWReportMessage](#) reports several kinds of messages: errors, warnings, and informational messages.

An error occurs when the contents of the plug-in's input prevents it from continuing its operation. Errors cause the IDE to cancel any

larger operations in progress. For example, if the user of the IDE selects **Debug** and a compiler or linker reports an error while building the project, the IDE will report the error, and cancel debugging of the target.

A warning occurs when the contents of the plug-in's input produces a result that the user might not expect or intend.

Informational messages do not indicate problems, but inform the user that something occurred. For example, the IDE reports informational messages when updating old target settings.

## Reporting Other User Messages

In previous versions of the API, plug-ins could use [CWAlert](#) to present a message to the user immediately in a modal dialog that required immediate action and user confirmation. Later versions of the API report this information in the message window, however. plug-ins should not use [CWAlert](#) to report general information to the user (instead use [CWReportMessage](#)).

## Displaying Plug-in Dialogs

On occasion, plug-ins may need to display their own dialogs, created and operated using operating system code, or perhaps other GUI libraries. Because the CodeWarrior IDE includes its own internal code for managing windows which does not know about such dialogs, it is important for plug-ins to notify the IDE before and after displaying such dialogs.

Before displaying its own dialog or alert, a plug-in should call [CWPreDialog](#). After closing the window, the plug-in should call [CWPostDialog](#). These calls inform the IDE of the dialog's creation and closure, and ensure that windowing operations work properly.

## Checking for User Breaks

A plug-in should regularly call [CWUserBreak](#) while performing an intensive operation. Calling [CWUserBreak](#) allows the IDE to perform other parts of its operation and checks to see if the user has issued a command to cancel the plug-in's operation.

Also, the IDE checks for user breaks while doing its own intensive operations. A call to the IDE that returns [cwErrUserCancelled](#) means that the user has cancelled the current operation.

If [CWUserBreak](#) or any other call to the IDE returns [cwErrUserCancelled](#), the plug-in should stop its processing immediately, clean up, and return control to the IDE.

## **Obsolete User Interface Routines**

Mac OS    Although they are obsolete, the IDE's compiler and linker plug-in API still provide `CWOSAlert` and `CWOSErrorMessage` routines to make it easier to port older plug-ins to the latest API. These routines are only available in the Mac OS hosted IDE.

## **Error Handling**

This section discusses API error handling, and error routines provided by the IDE.

The topics in this section are:

- [General Error Handling](#)
- [Obtaining Error Information](#)
- [Reporting Errors to the IDE](#)

### **General Error Handling**

The CodeWarrior API provides standard error result codes from most routines. It is important to check these for reliable operation.

The API headers provide a macro `CWSUCCESS` that is useful when testing for successful completion. This define insulates you from the platform-dependent nature of error codes.

### **Obtaining Error Information**

When an error occurs, a plug-in may need additional information about the error. You can obtain the most recent platform-specific OS error code that occurred during an IDE callback by calling [CWGetCallbackOSError](#). This routine returns the native OS error

code, if any, that caused the API callback to fail. This may be useful in determining how to handle an error.

## Reporting Errors to the IDE

All plug-in entry points obey standard plug-in calling conventions, specified by the `CWPlugin_ENTRY #define` appearing in `CWPlugins.h`:

```
#if CWPlugin_HOST == CWPlugin_HOST_MACOS
    #define CWPlugin_ENTRY(function_name)
                pascal short (function_name)
#elif CWPlugin_HOST == CWPlugin_HOST_WIN32
    #define CWPlugin_ENTRY(function_name)
                short (__stdcall function_name)
#endif
```

---

The `short` function result for all plug-in entry points returns errors to the IDE. This result should be one of the error codes defined in `CWPluginErrors.h`, or in `DropInPanelWind32.h` or `DropInPanel.h` for settings panels.

Before returning an error from its main entry point, a plug-in should first call [CWDonePluginRequest](#). The plug-in passes any error code it generated during the pending request back to the IDE. The IDE then returns an error code to the plug-in that should be passed back to the IDE as the `return` result of the main entry point.

Note that API error codes are simplified, logical error codes, rather than detailed OS-specific error codes. In some cases, these error codes may not tell the whole story, and your plug-in may wish to report an originating OS error code to the IDE, in addition to the value passed back to the IDE via [CWDonePluginRequest](#). Use [CWSetPluginOSError](#) to do this.

Mac OS	Plug-ins may use <a href="#">CWMacOSErrToCWResult</a> to convert OS errors into IDE errors. Note that this routine only handles the most common errors; if your plug-in generates unusual OS errors with frequency, you may prefer to map OS error codes to IDE error codes yourself for more meaningful results.
--------	---

# Plug-in API Reference

---

This chapter describes core services provided by the plug-in API, commonly used by most plug-ins.

## Overview

This chapter covers the following topics:

- [Routines for Plug-ins](#)
- [User Routines for Plug-ins](#)
- [Data Structures for Plug-ins](#)
- [Constants for Plug-ins](#)
- [Result Codes for Plug-ins](#)

## Routines for Plug-ins

This section documents core API routines available to all plug-ins.

### Plug-in Context

All routines provided by the CodeWarrior IDE require a value of type [CWPluginContext](#) to be passed as the first parameter. To avoid duplication, this parameter is only explained once in detail , rather than for every routine. See “[CWPluginContext” on page 172](#) and “[Main Entry Point context Parameter” on page 77](#) for more information.

### Alphabetical Routine Index

This section lists all common plug-in routines alphabetically.

- [CWAddProjectEntry](#)

## Plug-in API Reference

### Alphabetical Routine Index

---

- [CWAlert](#)
- [CWAllocateMemory](#)
- [CWAllocMemHandle](#)
- [CWCreateNewTextDocument](#)
- [CWDonePluginRequest](#)
- [CWFIndAndLoadFile](#)
- [CWFreeMemHandle](#)
- [CWFreeMemory](#)
- [CWGetAccessPathInfo](#)
- [CWGetAccessPathListInfo](#)
- [CWGetAccessPathSubdirectory](#)
- [CWGetAPIVersion](#)
- [CWGetCallbackOSError](#)
- [CWGetCOMApplicationInterface](#)
- [CWGetCOMProjectInterface](#)
- [CWGetCOMDesignInterface](#)
- [CWGetCOMTargetInterface](#)
- [CWGetFileInfo](#)
- [CWGetFileText](#)
- [CWGetFrameworkCount](#)
- [CWGetFrameworkInfo](#)
- [CWGetFrameworkSharedLibrary](#)
- [CWGetIDEInfo](#)
- [CWGetMemHandleSize](#)
- [CWGetNamedPreferences](#)
- [CWGetOutputFileDialog](#)
- [CWGetOverlay1FileInfo](#)
- [CWGetOverlay1GroupsCount](#)
- [CWGetOverlay1GroupInfo](#)
- [CWGetOverlay1Info](#)
- [CWGetPluginData](#)

- [CWGetPluginRequest](#)
- [CWGetProjectFile](#)
- [CWGetProjectFileCount](#)
- [CWGetSegmentInfo](#)
- [CWGetTargetDataDirectory](#)
- [CWGetTargetName](#)
- [CWLckMemHandle](#)
- [CWMacOSErrToCWResult](#)
- [CWPostDialog](#)
- [CWPostFileAction](#)
- [CWPrefFileAction](#)
- [CWPrefDialog](#)
- [CWReleaseFileDialog](#)
- [CWRemoveProjectEntry](#)
- [CWReportMessage](#)
- [CWResizeMemHandle](#)
- [CWResolveRelativePath](#)
- [CWSetModDate](#)
- [CWSetPluginOSError](#)
- [CWShowStatus](#)
- [CWStorePluginData](#)
- [CWUnlockMemHandle](#)
- [CWUserBreak](#)

## Functional Routine Index

This section lists all common plug-in routines grouped by function.

### Request Handling

- [CWDonePluginRequest](#)
- [CWGetPluginRequest](#)

## **Memory Management**

- [CWAllocateMemory](#)
- [CWAllocMemHandle](#)
- [CWFreeMemHandle](#)
- [CWFreeMemory](#)
- [CWGetMemHandleSize](#)
- [CWLockMemHandle](#)
- [CWResizeMemHandle](#)
- [CWUnlockMemHandle](#)

## **Plug-in Data**

- [CWGetPluginData](#)
- [CWStorePluginData](#)

## **Preference Data**

- [CWGetNamedPreferences](#)

## **File Management**

- [CWFindAndLoadFile](#)
- [CWGetFileInfo](#)
- [CWGetFileText](#)
- [CWPostFileAction](#)
- [CWPreFileAction](#)
- [CWReleaseFileText](#)
- [CWSetModDate](#)

## **Directory Information**

- [CWGetAccessPathInfo](#)
- [CWGetAccessPathListInfo](#)
- [CWGetAccessPathSubdirectory](#)
- [CWResolveRelativePath](#)

## **Project File Information**

- [CWAddProjectEntry](#)
- [CWGetProjectFile](#)
- [CWGetProjectFileCount](#)

## **Target Information**

- [CWGetTargetDataDirectory](#)
- [CWGetTargetName](#)
- [CWGetOutputFileDirectory](#)

## **Overlay Information**

- [CWGetOverlay1FileInfo](#)
- [CWGetOverlay1GroupInfo](#)
- [CWGetOverlay1GroupsCount](#)
- [CWGetOverlay1Info](#)

## **Segment Information**

- [CWGetSegmentInfo](#)

## **IDE Information**

- [CWGetAPIVersion](#)
- [CWGetIDEInfo](#)

## **User Interaction**

- [CWAlert](#)
- [CWCreatenewTextDocument](#)
- [CWPostDialog](#)
- [CWPreDialog](#)
- [CWReportMessage](#)
- [CWShowStatus](#)
- [CWUserBreak](#)

## **Error Handling**

- [CWGetCallbackOSError](#)

- [CWMacOSErrToCWResult](#)
- [CWSetPluginOSError](#)

### COM Object Interfaces

- [CWGetCOMApplicationInterface](#)
  - [CWGetCOMProjectInterface](#)
  - [CWGetCOMDesignInterface](#)
  - [CWGetCOMTargetInterface](#)
- 

**NOTE** The COM routines are for use by COM plug-ins which extend the CodeWarrior IDE, and are only minimally documented in this reference. COM routines do not work within compiler, linker, preference panel, or VCS plug-ins. See the *CodeWarrior COM API Reference* documentation for more information.

---

## CWAddProjectEntry

Description Adds a file to a project.

Prototype

```
#include <"DropinCompilerLinker.h">
CW_CALLBACK CWAddProjectEntry(
    CWPluginContext context,
    const CWFileSpec* fileSpec,
    Boolean isGenerated,
    const CWNewProjectEntryInfo* projectEntryInfo,
    long* whichfile);
```

Parameters

Parameters for this function are:

context	Private, opaque IDE state.
filespec	Specifies the file to add to the project.
isGenerated	When true, specifies that the file has been created for the current request. For example, a compiler that generates a C source code file and adds it to the project would pass true in isGenerated.

	projectEntryInfo	Specifies where to add the new file. Contains fields for flat, segmented, and overlayed linkers. The appropriate fields should be set for whichever model is used by the current linker. All position indexes are zero-based. If the pointer to this struct is NULL, the IDE will use default values for positioning the new file.
	whichfile	Returns the index at which the file was added, in current target link order.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .	
Remarks	Use CWAddProjectEntry to add a file to the project. This call is typically used by compilers that generate source code (for example, a C-to-assembly compiler).  If a plug-in calls CWAddProjectEntry during a make operation that is in progress, the IDE will restart the build operation.	
See Also	<a href="#">“CWFileSpec” on page 162</a> <a href="#">“CWNewProjectEntryInfo” on page 168</a> <a href="#">“CWSetModDate” on page 143</a> <a href="#">“Adding a File to a Project” on page 226</a> <a href="#">“CWRmRemoveProjectEntry” on page 139</a>	

## CWAlert

Description	Displays a message in the IDE's message window.
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWAlert( <a href="#">CWPluginContext</a> context, const char* msg1, const char* msg2, const char* msg3, const char* msg4);
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
msg1, msg2, msg3, msg4	Specifies the alert text, as C character strings. All four strings are concatenated in suffix order before display.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWAlert to add a message to the IDE’s message window. Previous versions of the IDE presented information in a modal alert box when a plug-in called this routine. The IDE now presents this information in a message window instead. New plug-ins should use <a href="#">CWReportMessage</a> instead of CWAlert.
See Also	<a href="#">“CWReportMessage” on page 140</a>

## **CWAllocateMemory**

Description      Allocates a block of memory referred to with a pointer.

Prototype    

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWAllocateMemory(
    CWPluginContext context,
    long size,
    Boolean isPermanent,
    void** ptr);
```

Parameters    Parameters for this function are:

context	Private, opaque IDE state.
size	Specifies the size of the block to allocate, in bytes.

<code>isPermanent</code>	Specifies the life of the allocation. If <code>isPermanent</code> is true, the IDE will keep the allocation between requests to the plug-in and even after the plug-in is unloaded from memory. Memory allocated with <code>isPermanent</code> set to true must be explicitly deallocated by the plug-in itself. If <code>isPermanent</code> is false, the memory is deallocated when the plug-in finishes handling the current request.
<code>ptr</code>	Returns a pointer to the allocated memory. The pointer is valid only if <code>CWAllocateMemory</code> returns <a href="#">cwNoErr</a> .
<b>Return</b>	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
<b>Remarks</b>	The pointer returned by this routine should be disposed by calling <a href="#">CWFreeMemory</a> . Memory allocated with this routine may be accessed simply by dereferencing the returned pointer.  The only reliable way to manage permanently allocated memory is to allocate it when the IDE makes a <a href="#">reqTargetLoaded</a> call to the plug-in, and to dispose it during a <a href="#">reqTargetUnloaded</a> call.
<b>See Also</b>	<a href="#">“CWFreeMemory” on page 113</a> <a href="#">“Managing Target Storage” on page 230</a>

## CWAllocMemHandle

<b>Description</b>	Allocates a <a href="#">CWMemHandle</a> , an indirectly referenced, resizable block of memory.
<b>Prototype</b>	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWAllocMemHandle(     <a href="#">CWPluginContext</a> context,     long size,     Boolean useTempMemory,     <a href="#">CWMemHandle</a>* handle);</pre>
<b>Parameters</b>	Parameters for this function are:

context	Private, opaque IDE state.
size	Specifies the size of the block to allocate, in bytes.
useTempMemory	Specifies whether the memory allocation should be from the temporary (process manager) heap or the application heap. Set useTempMemory to true to use temporary memory. This flag only affects Mac OS-hosted plug-ins.
handle	Returns the allocated memory in a <a href="#">CWMemHandle</a> .
Return	See “ <a href="#">Result Codes for Plug-ins</a> ” on page 193.
Remarks	The handle returned by this routine should be disposed by calling <a href="#">CWFreeMemHandle</a> . Memory allocated by this routine can only be referenced indirectly, by first calling <a href="#">CWLockMemHandle</a> . When access to a handle is no longer required, it should be unlocked using <a href="#">CWUnlockMemHandle</a> .
	Memory allocated with CWAllocMemHandle is deallocated when a plug-in calls <a href="#">CWFreeMemHandle</a> or when the plug-in finishes servicing a request. To allocate memory that persists between calls made by the IDE, use <a href="#">CWAallocateMemory</a> .
See Also	<a href="#">“CWAallocateMemory” on page 106</a> <a href="#">“CWFreeMemHandle” on page 112</a> <a href="#">“CWGetMemHandleSize” on page 124</a> <a href="#">“CWResizeMemHandle” on page 141</a> <a href="#">“CWLockMemHandle” on page 134</a> <a href="#">“CWUnlockMemHandle” on page 147</a>

## **CWCreateNewTextDocument**

Description	Opens a new text editor window containing specified text.
Prototype	#include "DropinCompilerLinker.h"

```
CW_CALLBACK CWCreateNewTextDocument(
    CWPluginContext           context,
    const CWNewTextDocumentInfo* docinfo);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	docinfo      Specifies the name and contents of the new editor window.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWCreateNewTextDocument to open a text editor window in which to present the textual result of an operation. A compiler or linker typically uses this routine to show disassembly and preprocessor text.
See Also	<a href="#">“CWNewTextDocumentInfo” on page 170</a>

## CWDonePluginRequest

Description	Tells the IDE that a plug-in has finished handling a request.
Prototype	#include "DropinCompilerLinker.h" <pre>CW_CALLBACK CWDonePluginRequest(     <u>CWPluginContext</u>   context,     <u>CWResult</u>        resultCode);</pre>
Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	resultCode     A value indicating whether the plug-in was able to handle a request successfully. Use <a href="#">cwNoErr</a> if the plug-in handled the request successfully. Use <a href="#">cwErrRequestFailed</a> if the plug-in wasn't able to complete the request successfully.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	When a plug-in has finished servicing a request, a plug-in must call CWDonePluginRequest to indicate success or failure to the IDE. In addition, the plug-in should pass the result code returned from

`CWDonePluginRequest` back to the IDE when returning from its `main()` function.

See Also [“cwNoErr” on page 197](#)

[“cwErrRequestFailed” on page 196](#)

## **CWFindAndLoadFile**

Description Locates and optionally loads the contents of a file and specifies a file dependency.

Prototype 

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWFindAndLoadFile(
    CWPluginContext context,
    const char* filename,
    CWFFileInfo* fileinfo);
```

Parameters Parameters for this function are:

`context` Private, opaque IDE state.

`filename` Specifies the name of the file to search for and load. If a file name appears more than once in a project, or its access paths, the first file found in a project's access paths having the specified name will be loaded.

`fileinfo` Specifies the file to load, and dependency information. On return, contains information about the file loaded.

Return See [“Result Codes for Plug-ins” on page 193.](#)

Remarks Use `CWFindAndLoadFile` to search for a file that is in one of the access paths listed in the **Access Paths** panel of the current target. This routine is typically used to load header and interface files that aren't necessarily in the project. If `CWFindAndLoadFile` successfully finds the file and optionally loads it into memory, it returns [cwNoErr](#).

`CWFindAndLoadFile` will attempt to load the requested file into memory, if the `suppressload` field in `fileinfo` is false. If `suppressload` is true, the IDE will not load the file into memory,

regardless of its type. CWFindAndLoadFile will only load a file into memory if it is a text file or cached precompiled header file.

When CWFindAndLoadFile finds a file and loads it into memory, it sets these fields in `fileinfo`:

- `filespec` contains a specification for the file
- `filedata` points to the file's contents in memory
- `filedatalength` contains the size of the file's contents, in bytes
- `filedatatype` will contain `cwFileTypeText` if the file contains text or `cwFileTypePrecompiledHeader` if the file is a cached precompiled header.

When CWFindAndLoadFile finds a file, but doesn't load it into memory, it sets these fields in `fileinfo`:

- `filespec` contains a specification for the file
- `filedata` contains `NULL`
- `filedatalength` contains `0`
- `filedatatype` will contain `cwFileTypeText` if the file contains text or `cwFileTypePrecompiledHeader` if the file is a cached precompiled header, or `cwFileTypeUnknown` for any other kind of file

---

**NOTE** Every call to CWFindAndLoadFile that returns (non-`NULL`) `filedata` must be balanced by a single call to [`CWReleaseFileText`](#).

---

The `isdependentoffile` and `dependencyType` fields of the [`CWFileInfo`](#) structure may be used to specify a source file dependency. If `dependencyType` is set to `cwNormalDependency` or `cwInterfaceDependency`, the IDE will know to recompile the file indicated by `isdependentoffile` when the interface or content of the file specified by `filespec` changes. If `dependencyType` is set to `cwNoDependency`, then no dependency relationship is inferred by the IDE.

`isdependentoffile` specifies which file depends upon the file being searched for, by index. `isdependentoffile` may be set to

`kCurrentCompiledFile` to specify that the file currently being compiled is the dependent file. Often, this is what you want.

**See Also**

- [“CWFileInfo” on page 159](#)
- [“CWDependencyType” on page 158](#)
- [“CWGetFileText” on page 121](#)
- [“CWCachePrecompiledHeader” on page 240](#)

## CWFreeMemHandle

**Description** Deallocates handles allocated with [CWAllocMemHandle](#).

**Prototype**

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWFreeMemHandle(
    CWPluginContext context,
    CWMemHandle handle);
```

**Parameters** Parameters for this function are:

- `context` Private, opaque IDE state.
- `handle` The [CWMemHandle](#) to deallocate.

**Return** See [“Result Codes for Plug-ins” on page 193](#).

**Remarks** To deallocate a [CWMemHandle](#) allocated with [CWAllocMemHandle](#), use `CWFreeMemHandle`. Handles should be disposed only once. Do not use `CWFreeMemHandle` to dispose blocks that are not handles.

---

**NOTE** All handles are automatically deallocated by when a plug-in returns to the IDE. However, you may wish to dispose large allocations sooner using `CWFreeMemHandle`.

---

**See Also**

- [“CWAllocMemHandle” on page 107](#)
- [“CWGetMemHandleSize” on page 124](#)
- [“CWResizeMemHandle” on page 141](#)
- [“CWLockMemHandle” on page 134](#)
- [“CWUnlockMemHandle” on page 147](#)

## CWFreeMemory

Description	Deallocates pointers allocated using <a href="#">CWAllocateMemory</a> .
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWFreeMemory( <a href="#">CWPluginContext</a> context, void* ptr, Boolean isPermanent);
Parameters	Parameters for this function are:  context                 Private, opaque IDE state. ptr                     The pointer block to deallocate. isPermanent             The value passed must match the value used when allocating the pointer with <a href="#">CWAllocateMemory</a> .
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	To free a block of memory allocated with <a href="#">CWAllocateMemory</a> , use CWFreeMemory. Pointers should be disposed only once. Do not use CWFreeMemory to dispose blocks that are not pointers.
<b>NOTE</b>	A plug-in should balance every call to <a href="#">CWAllocateMemory</a> with a single corresponding call to CWFreeMemory for the same pointer.
See Also	<a href="#">“CWAllocateMemory” on page 106</a> <a href="#">“Allocating and Disposing Handles” on page 84</a>

## CWGetAccessPathInfo

Description	Returns information about a single access path specified for the current project target.
Prototype	#include "CWPlugins.h" CW_CALLBACK CWGetAccessPathInfo( <a href="#">CWPluginContext</a> context, <a href="#">CWAccessPathType</a> pathType, long whichPath, <a href="#">CWAccessPathInfo</a> * pathInfo);

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	pathType      Set to one of <a href="#">cwSystemPath</a> or <a href="#">cwUserPath</a> . Specifies whether to return a user path or system path.
	whichPath      A zero-based index for the path to be returned.
	pathInfo      Returns information about the specified path.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	Plug-ins use CWGetAccessPathInfo to obtain information about the access paths specified in the <b>Access Paths</b> panel. By specifying an index that ranges from 0 to one less than the value returned for systemPathCount or userPathCount by <a href="#">CWGetAccessPathListInfo</a> , plug-ins may index all system and user paths (respectively) associated with the current project.
	The information returned in pathinfo includes the <a href="#">CWFfileSpec</a> for the access path, as well as a count of subdirectories and a flag indicating whether the IDE searches subdirectories of the path for project files or just the access path itself. If subdirectory searching is disabled, CWGetAccessPathInfo returns 0 for the subdirectory count, regardless of how many subdirectories actually exist in the access path.
	To obtain information about subdirectories of project access paths, use <a href="#">CWGetAccessPathSubdirectory</a> .
See Also	<a href="#">“CWGetAccessPathListInfo” on page 114</a> <a href="#">“CWGetAccessPathSubdirectory” on page 115</a> <a href="#">“CWAccessPathInfo” on page 154</a>

## CWGetAccessPathListInfo

Description	Returns information about the access paths specified for the current project target.
Prototype	<pre>#include "CWPlugins.h" CW_CALLBACK CWGetAccessPathListInfo(     CWPluginContext    context,</pre>

`CWAccessPathListInfo* pathListInfo);`

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	pathListInfo      Returns information about the access paths specified for the current project target, including the number of user and system paths.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	Plug-ins may determine the number of user and system paths in the current project target by calling <code>CWGetAccessPathListInfo</code> . This is useful when enumerating project paths using <a href="#">CWAccessPathInfo</a> .
See Also	<a href="#">“CWAccessPathListInfo” on page 155</a>

## **CWGetAccessPathSubdirectory**

Description	Returns information about a subdirectory of a project path.
Prototype	<code>#include "CWPlugins.h" CW_CALLBACK CWGetAccessPathSubdirectory(     <a href="#">CWPluginContext</a> context,     <a href="#">CWAccessPathType</a> pathType,     long whichPath,     long whichSubdirectory,     <a href="#">CWFileSpec</a>* subdirectory);</code>
Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	pathType      Set to one of <a href="#">cwSystemPath</a> or <a href="#">cwUserPath</a> . Specifies whether to return information about a user path or system path subdirectory.
	whichPath      A zero-based index specifying a path of type <code>pathType</code> in the current target.

---

whichSubdirectory	A zero-based index specifying a subdirectory of whichPath.
subdirectory	Returns a file specification for the requested access path subdirectory.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWGetAccessPathSubdirectory to obtain the subdirectories of any project path for which recursive project path searching is enabled. This is useful when passing a list of search paths to command line tools.
<b>NOTE</b>	CWGetAccessPathSubdirectory will not return subdirectories for access paths for which recursive path searching is disabled, regardless of how many actual subdirectories appear in the access path on disk. The small folder icon next to a path listed in <b>Edit &gt; Target Settings &gt; Access Paths</b> controls recursive searching. When the folder icon is visible, the IDE performs recursive searching.

---

To determine the range of valid indexes for whichPath, use [CWGetAccessPathListInfo](#). To determine the range of indexes for whichSubdirectory, use [CWGetAccessPathInfo](#).

See Also [“CWGetAccessPathInfo” on page 113](#)

[“CWGetAccessPathListInfo” on page 114](#)

## CWGetAPIVersion

Description	Returns the version of the plug-in API that the IDE is using to interact with the plug-in.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetAPIVersion(     <a href="#">CWPluginContext</a>    context,     long*                 version);</pre>
Parameters	Parameters for this function are:

	context	Private, opaque IDE state.
	version	Returns the version number of the API the IDE is using to interact with the plug-in.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>	
Remarks	Use CWGetAPIVersion at runtime to get the version of the plug-in API that the IDE is currently using to communicate with the plug-in. This may be useful in adapting to different IDE service levels.	
<b>NOTE</b>	The value returned by CWGetAPIVersion is not necessarily the latest API version supported by the IDE. The version reported will be less than the latest API version when the current plug-in requires an earlier API version, determined by the value of newestAPIVersion returned by its CWPlugin_GetDropInFlags entry point.	
See Also	<a href="#">“CWGetIDEInfo” on page 124</a> <a href="#">“DROPINCOMPILERLINKERAPIVERSION” on page 297</a>	

## CWGetCallbackOSSError

Description	Returns the error the host operating system returned to the IDE the last time the plug-in called the IDE.
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWGetCallbackOSSError( <a href="#">CWPluginContext</a> context, <a href="#">CWOSResult</a> * error);
Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	error        Returns the last error code reported by the host operating system during a call to the IDE by the plug-in.
Return	<a href="#">cwNoErr</a>
Remarks	Use this routine to get the error that the host operating system returned to the IDE when the IDE was last called by the plug-in.

This is most useful when an IDE routine reports a result of `cwErrOSError`.

See Also [“CWMacOSErrToCWResult” on page 136](#)

## **CWGetCOMApplicationInterface**

Description	Returns a reference to a COM interface for the IDE as an application object.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetCOMApplicationInterface(     CWPluginContext          context,     struct ICodeWarriorApp **app);</pre>
Parameters	Parameters for this function are:
	context         Private, opaque IDE state.
	app             Returns a reference to the CodeWarrior IDE application's COM interface. Can be used to obtain other properties and services of the IDE.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	This routine is part of the CodeWarrior Plug-in API but cannot be used by compilers, linkers, settings panels, or version control plug-ins. See <i>CodeWarrior COM API Reference</i> for further information.

## **CWGetCOMProjectInterface**

Description	Returns a reference to a COM interface for the current project.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetCOMProjectInterface(     CWPluginContext          context,     struct ICodeWarriorProject **project);</pre>
Parameters	Parameters for this function are:
	context         Private, opaque IDE state.
	project         Returns a reference to the current project's COM interface. Can be used to obtain properties and services of the current project.

Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	This routine is part of the CodeWarrior Plug-in API, but cannot be used by compilers, linkers, settings panels, or version control plug-ins. See <i>CodeWarrior COM API Reference</i> for further information.
See Also	n/a

## **CWGetCOMDesignInterface**

Description	Returns a reference to a COM interface for the design associated with the current target.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetCOMDesignInterface(     CWPluginContext           context,     struct ICodeWarriorDesign **design);</pre>
Parameters	Parameters for this function are:  context      Private, opaque IDE state. design        Returns a reference to the current target design's COM interface. Can be used to manipulate user interface controls.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	This routine is part of the CodeWarrior Plug-in API, but cannot be used by compilers, linkers, settings panels, or version control plug-ins. See <i>CodeWarrior COM API Reference</i> for further information.
See Also	n/a

## **CWGetCOMTargetInterface**

Description	Returns a reference to a COM interface for the current project target.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetCOMTargetInterface(     CWPluginContext           context,     struct ICodeWarriorTarget **target);</pre>
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
target	Returns a reference to the current target's COM interface. Can be used to obtain properties and services of the current target.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	This routine is part of the CodeWarrior Plug-in API, but cannot be used by compilers, linkers, settings panels, or version control plug-ins. See <i>CodeWarrior COM API Reference</i> for further information.
See Also	n/a

## **CWGetFileInfo**

Description	Returns information about a file in the active project.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetFileInfo(     <a href="#">CWPluginContext</a>      context,     long                  whichfile,     Boolean               checkFileLocation,     <a href="#">CWProjectFileInfo</a>* fileinfo);</pre>
Parameters	Parameters for this function are:
	context                      Private, opaque IDE state.
	whichfile                    The number of the file in the active target to get information for.
	checkFileLocation           Set to true to request that the IDE verify the location of the file returned in fileinfo. Set to false to request the most recently cached location for the file.
	fileinfo                     Returns information about the file.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWGetFileInfo to get information about a file in the active project target, such as its file specification, its modification date, and many other properties of the file, and how the IDE processes the file during build operations.

The `whichfile` argument specifies the index of the file for which to return information, in the link order of the active project target. The files are numbered from 0 to the value returned by [CWGetProjectFileCount](#) - 1.

`checkFileLocation` is typically used when iterating over all the files in a project multiple times. Setting this flag to `true` tells the IDE to use cached locations for files, rather than verifying a file's location with the host operating system. In some cases, this may improve performance. Usually, most plug-ins will specify `false` for this parameter, which tells the IDE to verify a file's true location on disk each time `CWGetFileInfo` is called.

---

**NOTE** The definition of the `fileinfo` structure returned by `CWGetFileInfo` is platform dependent.

---

**Versions** In versions of the API prior to 11, the `objmoddate` field of the [CWProjectFileInfo](#) structure returned by this routine reported the modification date of the associated source file, rather than its object code.

**See Also** [“CWProjectFileInfo” on page 174](#)

[“CWGetProjectFileCount” on page 132](#)

## CWGetFileText

**Description** Loads the contents of a file.

**Prototype**

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWGetFileText(
    CWPluginContext context,
    const CWFileSpec* filespec,
    const char** text,
    long* textLength,
    short* filedatatype);
```

**Parameters** Parameters for this function are:

<code>context</code>	Private, opaque IDE state.
<code>filespec</code>	The file specification of the file to load.
<code>text</code>	Returns a pointer to the contents of the file.

textLength	Returns the length of the file's contents in bytes.
filedatatype	Returns the file's type. Values for this argument are <a href="#">cwFileTypePrecompiledHeader</a> , <a href="#">cwFileTypeText</a> , <a href="#">cwFileTypeUnknown</a>
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	Use <code>CWGetFileText</code> to retrieve the contents of a file. If the file is a text file that is opened in an editor window, the IDE returns a pointer in <code>text</code> to the contents of the open editor window. If the file is on disk and is not open in an editor window, the IDE loads the contents of the file and returns a pointer to that data in <code>text</code> .
<b>NOTE</b>	Every call to <code>CWGetFileText</code> that returns (non-NULL) <code>text</code> must be balanced by a single call to <a href="#">CWReleaseFileText</a> .
<b>NOTE</b>	The file text returned by <code>CWGetFileText</code> is declared to be constant. plug-ins should not modify the text data returned in <code>text</code> .
See Also	<a href="#">“CWFindAndLoadFile” on page 110</a> <a href="#">“CWReleaseFileText” on page 139</a>

## **CWGetFrameworkCount**

Description	Gets the number of frameworks.
Prototype	<code>CW_CALLBACK CWGetFrameworkCount(</code> <code>CWPPluginContext context,</code> <code>long* frameworkCount);</code>
Parameters	Parameters for this function are:  <code>context</code> Private, opaque IDE state. <code>frameworkCount</code> Returns the number of frameworks.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
See Also	<a href="#">“CWGetFrameworkInfo” on page 123</a>

[“CWGetFrameworkSharedLibrary” on page 123](#)

## CWGetFrameworkInfo

Description	Gets information about a framework, as a <a href="#">CWFrameworkInfo</a> structure.
Prototype	<code>CW_CALLBACK CWGetFrameworkInfo(     <a href="#">CWPluginContext</a> context,     long whichFramework,     CWframeworkInfo* frameworkInfo);</code>
Parameters	Parameters for this function are:
	context                          Private, opaque IDE state.
	whichFramework                 The framework about which to get information.
	frameworkInfo                 Returns the framework information.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
See Also	<a href="#">“CWGetFrameworkCount” on page 122</a> <a href="#">“CWGetFrameworkSharedLibrary” on page 123</a> <a href="#">“CWFrameworkInfo” on page 165</a>

## CWGetFrameworkSharedLibrary

Description	Gets the shared library for a framework.
Prototype	<code>CW_CALLBACK CWGetFrameworkSharedLibrary(     <a href="#">CWPluginContext</a> context,     long whichFramework,     CWFfileSpec* frameworkSharedLibrary);</code>
Parameters	Parameters for this function are:
	context                          Private, opaque IDE state.
	whichFramework                 The framework about which to get information.
	frameworkSharedLibrary         Returns the shared library used by the framework.

Return	See “ <a href="#">Result Codes for Plug-ins</a> ” on page 193.
See Also	<a href="#">“CWGetFrameworkCount” on page 122</a> <a href="#">“CWGetFrameworkInfo” on page 123</a>

## **CWGetIDEInfo**

Description	This returns version information about the IDE and which version of the dropin API it implements.
Prototype	#include "CWPlugins.h" CW_CALLBACK CWGetIDEInfo( <a href="#">CWPluginContext</a> context, <a href="#">CWIDEInfo</a> * info);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. info             Returns information about the IDE, including major, minor, bug fix, and build version numbers, as well as the highest API version supported by the IDE.
Return	See “ <a href="#">Result Codes for Plug-ins</a> ” on page 193.
Remarks	Note that the version of the dropin API returned by this call may be different from the one returned by <a href="#">CWGetAPIVersion</a> . <a href="#">CWGetAPIVersion</a> returns the version of the API that the IDE is providing to a particular plug-in. This can be less than the IDE's latest supported dropin API version (reported by this call) if a plug-in is written to use an older API.
See Also	<a href="#">“CWGetAPIVersion” on page 116</a> <a href="#">“CWIDEInfo” on page 165</a>

## **CWGetMemHandleSize**

Description	Returns the size of a <a href="#">CWMemHandle</a> .
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWGetMemHandleSize( <a href="#">CWPluginContext</a> context, <a href="#">CWMemHandle</a> handle,

```
long* size);
```

Parameters	Parameters for this function are:
	context Private, opaque IDE state.
	handle The handle whose size is to be determined.
	size Returns the size of the handle, in bytes.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	To get the size of a block of memory allocated with <a href="#">CWAllocMemHandle</a> or resized by <a href="#">CWResizeMemHandle</a> , use CWGetMemHandleSize.
See Also	<a href="#">“CWAllocMemHandle” on page 107</a> <a href="#">“CWFreeMemHandle” on page 112</a> <a href="#">“CWResizeMemHandle” on page 141</a> <a href="#">“CWLockMemHandle” on page 134</a> <a href="#">“CWUnlockMemHandle” on page 147</a>

## **CWGetNamedPreferences**

Description	Returns the settings data of a settings panel.
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWGetNamedPreferences( <a href="#">CWPluginContext</a> context, const char* prefsname, <a href="#">CWMemHandle</a> * prefsdata);
Parameters	Parameters for this function are:
	context Private, opaque IDE state.
	prefsname The name of the settings panel whose preferences are to be loaded, specified as a C character string.
	prefsdata Returns the preference data in a handle.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>

Remarks	<p>The IDE returns a <a href="#">CWMemHandle</a> in prefsdata containing the settings data for the settings panel specified by prefsname.</p> <p>The IDE has no knowledge of the contents and format of the preference data. Only the plug-in and its associated settings panel plug-ins manage and interpret the contents of the settings data.</p> <p>The handle returned becomes the plug-in's property and should be disposed with <a href="#">CWFreeMemHandle</a> when no longer needed. Prior to accessing the handle contents, the handle should be locked using <a href="#">CWLokMemHandle</a>.</p>
See Also	<p><a href="#">“CWMemHandle” on page 166</a></p> <p><a href="#">“CWFreeMemHandle” on page 112</a></p> <p><a href="#">“CWLokMemHandle” on page 134</a></p>

## **CWGetOutputFileDialog**

Description	Returns the folder specified in the <b>Output Directory</b> field of the <b>Target Settings</b> panel.				
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetOutputFileDialog(     <a href="#">CWPluginContext</a> context,     <a href="#">CWFileSpec</a>* outputFileDirectory);</pre>				
Parameters	Parameters for this function are:				
	<table><tr><td>context</td><td>Private, opaque IDE state.</td></tr><tr><td>outputFileDirectory</td><td>Returns the file specification of the output directory.</td></tr></table>	context	Private, opaque IDE state.	outputFileDirectory	Returns the file specification of the output directory.
context	Private, opaque IDE state.				
outputFileDirectory	Returns the file specification of the output directory.				
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .				
Remarks	Use <code>CWGetOutputFileDialog</code> to determine the location where linker output should be stored, as specified by the user in the <b>Target Settings</b> panel.				
See Also	<a href="#">“CWGetProjectFile” on page 131</a>				

## **CWGetOverlay1FileInfo**

Description	Returns information about a file in an overlay.
-------------	---

Prototype    `#include "DropinCompilerLinker.h"  
CW_CALLBACK CWGetOverlay1FileInfo(  
    CWPluginContext context,  
    long whichgroup,  
    long whichoverlay,  
    long whichoverlayfile,  
    CWOverlay1FileInfo* fileinfo);`

Parameters    Parameters for this function are:

context	Private, opaque IDE state.
whichgroup	Specifies the overlay group number.
whichoverlay	Specifies the overlay number.
whichoverlayfile	Specifies the overlay file number.
fileinfo	Returns information about the file in the overlay.

Return    See [“Result Codes for Plug-ins” on page 193](#).

Remarks    Use `CWGetOverlay1FileInfo` to get an overlay file's position within a project. Each of the `whichgroup`, `whichoverlay`, and `whichoverlayfile` indexes is zero based, and specifies the overlay component for which to return information. The `fileinfo` result specifies the position in the file list of the current target

---

**NOTE**    Currently, overlays are only used for certain embedded targets (but information about project overlays can be accessed through the API on any host platform).

---

See Also

[“CWGetOverlay1Info” on page 129](#)

[“Getting Information About Overlays and Segments” on page 89](#)

[“Specifying Project Components” on page 87](#)

[“Project Structure” on page 86](#)

## **CWGetOverlay1GroupInfo**

Description    Returns information about an overlay group in the project.

Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetOverlay1GroupInfo(     <u>CWPluginContext</u>      context,     long                  whichgroup,     <u>CWOv</u><u>erlay1GroupInfo</u>* groupinfo);</pre>
Parameters	Parameters for this function are:  context      Private, opaque IDE state. whichgroup    The overlay group to get information for. groupinfo     Returns information about the overlay group.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWGetOverlay1GroupInfo to get the name, the address, and the number of overlays in an overlay group. This routine is useful when iterating through overlay groups.  Overlay groups are specified by index. The index of the first overlay group is 0. The index of the last is <a href="#">CWGetOverlay1GroupsCount - 1</a> .
See Also	<a href="#">“CWOv</a> <a href="#">erlay1GroupInfo” on page 171</a>

## **CWGetOverlay1GroupsCount**

Description	Returns the number of overlay groups.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetOverlay1GroupsCount(     <u>CWPluginContext</u>      context,     long*                  count);</pre>
Parameters	Parameters for this function are:  context      Private, opaque IDE state. count        Returns the number of overlay groups in the project.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWGetOverlay1GroupsCount to get the number of overlay groups in the active project. This call is useful when iterating through overlay groups.

See Also [“CWGetOverlay1GroupInfo” on page 127](#)

## CWGetOverlay1Info

Description Returns information about an overlay in a group.

Prototype

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWGetOverlay1Info(
    CWPluginContext context,
    long whichgroup,
    long whichoverlay,
    CWOvleray1Info* overlayinfo);
```

Parameters Parameters for this function are:

context	Private, opaque IDE state.
whichgroup	The overlay group number.
whichoverlay	The overlay number.
overlayinfo	Returns information about the overlay.

Return See [“Result Codes for Plug-ins” on page 193](#).

Remarks Use CWGetOverlay1Info to get the name and number of files in an overlay. This routine is useful when iterating through overlays.

Overlays are specified by index. The index of the first overlay is 0. The index of the last is numoverlays - 1, where numoverlays is the value returned by [CWGetOverlay1GroupInfo](#).

See Also [“CWGetOverlay1FileInfo” on page 126](#)

[“CWGetOverlay1GroupInfo” on page 127](#)

## CWGetPluginData

Description Returns plug-in data associated with a file in the active project.

Prototype

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWGetPluginData(
    CWPluginContext context,
    long whichfile,
    CWDataType type,
    CWMemHandle* pluginData);
```

---

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	whichfile     Specifies the file to retrieve data for. To obtain global data (that is, data for the target, not a specific target file), specify <a href="#">kTargetGlobalPluginData</a> .
	type          Specifies which kind of data to retrieve.
	pluginData    Returns a <a href="#">CWMemHandle</a> containing the data associated with the file. Returns NULL if no data of type type exists.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	CWGetPluginData retrieves data in the project for a file, whichfile, having type type. The format of this data is maintained and understood only by the plug-in, not the IDE. The data stored is maintained on disk, and retained with a project across invocations of the IDE, unlike “target storage,” accessed using <a href="#">CWGetTargetStorage</a> and <a href="#">CWSetTargetStorage</a> .  The handle returned becomes the plug-in’s property and should be disposed with <a href="#">CWFreeMemHandle</a> when no longer needed. Prior to accessing the handle contents, the handle should be locked using <a href="#">CWLockMemHandle</a> .
See Also	<a href="#">“CWStorePluginData” on page 145</a> <a href="#">“CWGetTargetStorage” on page 256</a> <a href="#">“CWSetTargetStorage” on page 266</a> <a href="#">“CWFreeMemHandle” on page 112</a> <a href="#">“CWLockMemHandle” on page 134</a> <a href="#">“Storing and Retrieving Plug-in Data” on page 91</a>

## **CWGetPluginRequest**

Description	Returns a value indicating the task the IDE is currently asking the plug-in to perform.
-------------	---

Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetPluginRequest(     <u>CWPluginContext</u> context,     long* request);</pre>
Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	request     Returns a request code indicating the service the IDE is requesting from the plug-in. Returns <a href="#">reqInitialize</a> , <a href="#">reqTerminate</a> , or some other value.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	When called by the IDE, a plug-in should call <code>CWGetPluginRequest</code> to determine what the IDE is asking the plug-in to do.
See Also	<a href="#">“reqInitialize” on page 192</a> <a href="#">“reqTerminate” on page 193</a> <a href="#">“Basic Request Handling” on page 75</a> <a href="#">“Handling Compiler and Linker Requests” on page 211</a> <a href="#">“Settings Panel Requests” on page 345</a>

## CWGetProjectFile

Description	Returns the project's file specification.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetProjectFile(     <u>CWPluginContext</u> context,     <u>CWFfileSpec</u>* projectSpec);</pre>
Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	projectSpec   Returns the file specification for the project.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .

## Plug-in API Reference

### *CWGetProjectFileCount*

---

Remarks	Use CWGetProjectFile to get a file specification for the active project. This routine is useful for getting information like the volume or folder where the project is stored.
Versions	In versions of the API prior to 11, instead of returning the <a href="#">CWFileSpec</a> for the project file, CWGetProjectFile returned the file specification for the target file.
See Also	<a href="#">“CWGetOutputFileDirectory” on page 126</a>

## **CWGetProjectFileCount**

Description	Returns the number of files in the current project target.
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWGetProjectFileCount( <a href="#">CWPluginContext</a> context, long* count);
Parameters	Parameters for this function are:  context       Private, opaque IDE state. count         Returns the number of files in the active project.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWGetProjectFileCount to get the number of files in the active project target. This call is useful when iterating through project files.
See Also	<a href="#">“CWGetFileInfo” on page 120</a>

## **CWGetSegmentInfo**

Description	Returns information about a segment in the project.
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWGetSegmentInfo( <a href="#">CWPluginContext</a> context, long whichsegment, <a href="#">CWProjectSegmentInfo</a> * segmentinfo);
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
whichsegment	Specifies a segment by index. Indexes are zero based, and match the order specified in the Segments tab of a project.
segmentinfo	Returns information about the specified segment.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWGetSegmentInfo to get information about a segment in the active project. The information returned specifies the segment name, and the segment’s resource attributes.
<b>NOTE</b>	Currently, segments are only used for Mac OS 68K targets (but segments can be used and accessed through the IDE on any host platform).

---

See Also [“CWProjectSegmentInfo” on page 178](#)

## CWGetTargetDataDirectory

Description	Returns the directory where files in the current project target are stored.				
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWGetTargetDataDirectory(     CWPluginContext context,     CWFileSpec* targetDataDirectorySpec);</pre>				
Parameters	Parameters for this function are:				
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;">context</td> <td>Private, opaque IDE state.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">targetDataDirectorySpec</td> <td>Returns the directory where data for the current target should be stored.</td> </tr> </table>	context	Private, opaque IDE state.	targetDataDirectorySpec	Returns the directory where data for the current target should be stored.
context	Private, opaque IDE state.				
targetDataDirectorySpec	Returns the directory where data for the current target should be stored.				
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>				
Remarks	CWGetTargetDataDirectory returns the target data directory for the current target and project. This is the directory where plug-ins should store additional temporary data files associated with a target, if necessary.				

---

This routine is most useful when implementing command line adapters. Most native plug-ins should use [CWGetTargetStorage](#) and [CWSetTargetStorage](#) to store per-target data instead of storing data files in the target directory manually.

**NOTE** Files in this directory will be deleted when the user chooses **Project > Remove Object Code**.

---

See Also [“CWGetProjectFile” on page 131](#)

## **CWGetTargetName**

Description Returns the name of the active target in the active project.

Prototype

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWGetTargetName(
    CWPluginContext context,
    char*           name,
    short            maxLength);
```

Parameters Parameters for this function are:

context	Private, opaque IDE state.
name	Returns the target's name as a C string.
maxLength	Specifies the space available in name for storing the target name, in bytes.

Return See [“Result Codes for Plug-ins” on page 193.](#)

Remarks Use CWGetTargetName to get the name of the active target in the current project.

See Also [“CWGetProjectFile” on page 131](#)

## **CWLockMemHandle**

Description Dereferences a handle block allocated with [CWMemHandle](#).

Prototype

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWLockMemHandle(
    CWPluginContext context,
    CWMemHandle     handle,
```

```
Boolean          moveHi,  
void**           ptr);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	handle        The plug-in passes a handle to dereference in this argument.
	moveHi        Specifies whether to move the handle to the top of the heap before locking it. If <code>moveHi</code> is true, the allocated memory is moved to the top of the heap, potentially reducing heap fragmentation. This flag only has effect on Mac OS-hosted plug-ins.
	ptr           Returns a pointer which may be dereferenced to access the handle memory. The pointer is valid only as long as the handle remains locked.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	To access the memory described by a <code>CWMemHandle</code> allocated with <code>CWAllocMemHandle</code> or resized with <code>CWResizeMemHandle</code> , use <code>CWLockMemHandle</code> .  When finished accessing a handle block, a plug-in should call <code>CWUnlockMemHandle</code> . On some platforms, locked handles may adversely affect memory management. Unlock handles as soon as possible after use.  Calls to <code>CWLockMemHandle</code> and <code>CWUnlockMemHandle</code> are counted. A handle remains locked as long as its lock count is greater than zero. When initially allocated, handles are unlocked. <code>CWLockMemHandle</code> increases a handle’s lock count, and <code>CWUnlockMemHandle</code> decreases its lock count.
See Also	<a href="#">“CWAllocMemHandle” on page 107</a> <a href="#">“CWFrmMemHandle” on page 112</a> <a href="#">“CWGetMemHandleSize” on page 124</a> <a href="#">“CWResizeMemHandle” on page 141</a>

[“CWUnlockMemHandle” on page 147](#)

## **CWMacOSErrToCWResult**

Description	Converts a Mac OS error code to a <a href="#">CWResult</a> error code.
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWMacOSErrToCWResult( <a href="#">CWPluginContext</a> context, OSerr               err);
Parameters	Parameters for this function are:  context      Private, opaque IDE state. err          Specifies the Mac OS error code to convert to an IDE error code.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	Use this utility routine to convert an error code returned by a call to a Mac OS operating system routine into a value recognized by the IDE. This routine always succeeds. The function result contains the IDE error code equivalent to the Mac OS error code passed in <code>err</code> .
<b>NOTE</b>	Use of this routine is discouraged. In its place, use your own code to convert OS errors to IDE errors.
Mac OS	This routine is available on Mac OS hosts only.
See Also	<a href="#">“Result Codes for Plug-ins” on page 193</a>

## **CWPostDialog**

Description	Informs the IDE that a plug-in has finished displaying a dialog.
Prototype	#include "CWPlugins.h" CW_CALLBACK CWPostDialog(CWPluginContext context);
Parameters	Parameters for this function are:  context      Private, opaque IDE state.
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .

Remarks	Call <code>CWPostDialog</code> after displaying your own dialog within a plug-in. This routine helps the IDE's internal window manager operate properly with your dialogs. Be sure to call <a href="#">CWPreDialog</a> before displaying the dialog.
See Also	<a href="#">“CWPreDialog” on page 138</a> <a href="#">“Displaying Plug-in Dialogs” on page 96</a>

## **CWPostFileAction**

Description	Informs the IDE that a plug-in has relinquished access to a file, allowing the IDE to reopen and reload the file, if necessary.				
Prototype	<pre>#include "CWPlugins.h" CW_CALLBACK CWPostFileAction     (CWPluginContext context, const CWFfileSpec      *theFile);</pre>				
Parameters	Parameters for this function are:  <table><tr><td>context</td><td>Private, opaque IDE state.</td></tr><tr><td>theFile</td><td>Specifies the file the plug-in has finished accessing.</td></tr></table>	context	Private, opaque IDE state.	theFile	Specifies the file the plug-in has finished accessing.
context	Private, opaque IDE state.				
theFile	Specifies the file the plug-in has finished accessing.				
Returns	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .				
Remarks	A plug-in should call <code>CWPostFileAction</code> after opening, reading, writing, or otherwise modifying any file in the current project for which <a href="#">CWPreFileAction</a> was previously called. This routine tells the IDE that the plug-in has relinquished control over the file, and that it may be reopened if the user has the file open in the IDE. A plug-in should call <a href="#">CWPreFileAction</a> before accessing the file. This is only necessary when accessing files directly, rather than through the plug-in API.				
NOTE	Currently, this routine is only available to version control plug-ins. This may change in the future.				
See Also	<a href="#">“CWPreFileAction” on page 138</a>				

## **CWPreDialog**

Description	Informs the IDE that a plug-in is about to display a dialog.
Prototype	#include "CWPlugins.h" CW_CALLBACK CWPreDialog (CWPluginContext context);
Parameters	Parameters for this function are:  context      Private, opaque IDE state.
Returns	See <a href="#">"Result Codes for Plug-ins" on page 193.</a>
Remarks	Call CWPreDialog prior to displaying your own dialog in a plug-in. This routine helps the IDE's internal window manager operate properly with your dialogs. Be sure to call <a href="#">CWPostDialog</a> after displaying the dialog.
See Also	<a href="#">"CWPostDialog" on page 136</a> <a href="#">"Displaying Plug-in Dialogs" on page 96</a>

## **CWPreFileAction**

Description	Informs the IDE that a plug-in needs access to a file that the IDE may have open.
Prototype	#include "CWPlugins.h" CW_CALLBACK CWPreFileAction (CWPluginContext context, const CWFfileSpec *theFile);
Parameters	Parameters for this function are:  context      Private, opaque IDE state.  theFile       Specifies the file the plug-in would like to access.
Returns	See <a href="#">"Result Codes for Plug-ins" on page 193.</a>
Remarks	A plug-in should call CWPreFileAction when it needs to read or write to the file specified by theFile. This tells the IDE that the plug-in needs access to the specified file, and that it should be closed. A plug-in should also call <a href="#">CWPostFileAction</a> after accessing the file.

<b>NOTE</b>	Currently, this routine is only available to version control plug-ins. This may change in the future.
-------------	---

See Also [“CWPostFileAction” on page 137](#)

## CWReleaseFileText

Description	Disposes the contents of a file retrieved with <a href="#">CWGetFileText</a> or <a href="#">CWFindAndLoadFile</a> .
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWReleaseFileText( <a href="#">CWPluginContext</a> context, const char* text);
Parameters	Parameters for this function are:  context      Private, opaque IDE state.  text          A pointer to the contents of a file previously loaded by <a href="#">CWGetFileText</a> or <a href="#">CWFindAndLoadFile</a> .
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	Use CWReleaseFileText to free the contents of a file retrieved using <a href="#">CWGetFileText</a> or <a href="#">CWFindAndLoadFile</a> .

<b>NOTE</b>	Every call to <a href="#">CWGetFileText</a> or <a href="#">CWFindAndLoadFile</a> which returns a non-NULL text or filedata pointer must be balanced by a corresponding call to <a href="#">CWReleaseFileText</a> .
-------------	--

See Also [“CWGetFileText” on page 121](#)

[“CWFindAndLoadFile” on page 110](#)

## CWRemoveProjectEntry

Description	Removes a file from the current target.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWRemoveProjectEntry( <a href="#">CWPluginContext</a> context, const CWFfileSpec* fileSpec)

---

Parameters	Parameters for this function are:
	context                         Private, opaque IDE state.
	fileSpec                         The file that needs to be removed.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use <code>CWRemoveProjectEntry</code> to remove a file from the project.  If a plug-in calls <code>CWRemoveProjectEntry</code> during a make operation that is in progress, the IDE will restart the build operation.
See Also	<a href="#">“CWFileSpec” on page 162</a> <a href="#">“Removing a File from a Project” on page 227</a>

## **CWReportMessage**

Description	Displays an error, warning, or informational message in the IDE's error and warnings window.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWReportMessage(     <u>CWPPluginContext</u>      context,     const <u>CWMessageRef</u>* msgRef,     const char *            line1,     const char *            line2,     short                   errorlevel,     long                    errorNumber);</pre>
Parameters	Parameters for this function are:
	context                         Private, opaque IDE state.
	msgRef                          Specifies the source text that generated the message. Pass NULL for informational messages.
	line1, line2                    Message text specified as C character strings. Each string appears on a separate line in the message window.

	errorlevel	Specifies the kind of message. Valid values are <a href="#">messagetypeInfo</a> , <a href="#">messagetypeWarning</a> , and <a href="#">messagetypeError</a> .
	errorNumber	This argument is reserved for future use.
Return	See “ <a href="#">Result Codes for Plug-ins</a> ” on page 193.	
Remarks	<p>To display an informational message using CWReportMessage, specify <a href="#">messagetypeInfo</a> for errorlevel, and pass NULL for msgRef. If you instead pass a pointer to a <a href="#">CWMessageref</a>, the message will be displayed as a warning. line1 and line2 will be displayed on separate lines.</p> <p>To display an error or warning using CWReportMessage, specify <a href="#">messagetypeError</a> or <a href="#">messagetypeWarning</a> respectively, and fill in msgRef. line1 should contain a general error description (for example “syntax error”), and line2 should contain an excerpt of the source in which the specific token generating the error or warning will be underlined using the offsets specified in msgRef.</p>	
<b>NOTE</b>	Code that previously used <a href="#">CWAlert</a> should be converted to use CWReportMessage.	

See Also	<a href="#">“CWAlert” on page 105</a>
	<a href="#">“CWShowStatus” on page 145</a>

## CWResizeMemHandle

Description	Changes the size of a <a href="#">CWMemHandle</a> .
Prototype	#include "DropinCompilerLinker.h" CW_CALLBACK CWResizeMemHandle( <a href="#">CWPluginContext</a> context, <a href="#">CWMemHandle</a> handle, long newSize);
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
handle	Specifies the handle to resize.
newSize	Specifies the new handle size, in bytes.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	Use CWResizeMemHandle to change the size of a handle created with <a href="#">CWAllocMemHandle</a> . CWResizeMemHandle supports resizing a handle to a smaller or larger size. Zero is a valid handle size.
See Also	<a href="#">“CWAllocMemHandle” on page 107</a> <a href="#">“CWFreeMemHandle” on page 112</a> <a href="#">“CWGetMemHandleSize” on page 124</a> <a href="#">“CWLockMemHandle” on page 134</a> <a href="#">“CWUnlockMemHandle” on page 147</a>

## **CWResolveRelativePath**

Description	Determines the absolute location of a path specified relative to another.				
Prototype	<pre>#include "CWPlugins.h" CW_CALLBACK CWResolveRelativePath(     <a href="#">CWPluginContext</a>           context,     const <a href="#">CWRelativePath</a>*   relativePath,     <a href="#">CWFfileSpec</a>*         fileSpec,     Boolean                   create);</pre>				
Parameters	<p>Parameters for this function are:</p> <table><tr><td style="vertical-align: top;">context</td><td>Private, opaque IDE state.</td></tr><tr><td style="vertical-align: top;">relativePath</td><td>A pointer to a <code>CWRelativePath</code> structure specifying the relative path to resolve.</td></tr></table>	context	Private, opaque IDE state.	relativePath	A pointer to a <code>CWRelativePath</code> structure specifying the relative path to resolve.
context	Private, opaque IDE state.				
relativePath	A pointer to a <code>CWRelativePath</code> structure specifying the relative path to resolve.				

	<b>fileSpec</b>	Returns a <code>CWFileSpec</code> specifying the absolute location described by the relative path.
	<b>create</b>	If the relative path specified by <code>relativePath</code> is not found, and this parameter is set to true, the path will be created.
<b>Return</b>	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .	
<b>Remarks</b>	Use <code>CWResolveRelativePath</code> to resolve a relative path to a full path. This is useful when determining the absolute location of a relative path stored in project preferences, typically obtained by calling <a href="#">CWPanelChooseRelativePath</a> .	
<b>See Also</b>	<a href="#">“CWRelativePath” on page 178</a> <a href="#">“CWPanelChooseRelativePath” on page 420</a>	

## CWSetModDate

<b>Description</b>	Sets the internal modification date of a project file.								
<b>Prototype</b>	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWSetModDate(     CWPluginContext context,     const CWFileSpec* filespec,     CWFileTime* moddate,     Boolean isGenerated);</pre>								
<b>Parameters</b>	Parameters for this function are:								
	<table> <tr> <td><code>context</code></td><td>Private, opaque IDE state.</td></tr> <tr> <td><code>filespec</code></td><td>Specifies which file's modification date to set.</td></tr> <tr> <td><code>moddate</code></td><td>Specifies the date and time that the file was modified.</td></tr> <tr> <td><code>isGenerated</code></td><td>Pass true in this argument to tell the IDE that the file has been created during the current request. For example, a compiler that generates a C source code as its output would pass true in <code>isGenerated</code>.</td></tr> </table>	<code>context</code>	Private, opaque IDE state.	<code>filespec</code>	Specifies which file's modification date to set.	<code>moddate</code>	Specifies the date and time that the file was modified.	<code>isGenerated</code>	Pass true in this argument to tell the IDE that the file has been created during the current request. For example, a compiler that generates a C source code as its output would pass true in <code>isGenerated</code> .
<code>context</code>	Private, opaque IDE state.								
<code>filespec</code>	Specifies which file's modification date to set.								
<code>moddate</code>	Specifies the date and time that the file was modified.								
<code>isGenerated</code>	Pass true in this argument to tell the IDE that the file has been created during the current request. For example, a compiler that generates a C source code as its output would pass true in <code>isGenerated</code> .								

---

Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>
Remarks	<p>Plug-ins should call <code>CWSetModDate</code> after modifying a file in the current project. This informs the IDE that a file has changed, and allows it to update its internal dependency tracking information. Note that <code>CWSetModDate</code> does not actually change a file's modification date, but rather the IDE's internal understanding of when a file was modified.</p> <p>Most plug-ins should set the <code>isGenerated</code> flag. The flag should be set if the plug-in generated the contents of the file in question. <code>isGenerated</code> should be set false for a file used by the project but not generated by the plug-in which may have changed. This is an unusual case.</p>
See Also	<a href="#">“CWFileSpec” on page 162</a> <a href="#">“CWFileType” on page 164</a> <a href="#">“CWGetProjectFile” on page 131</a> <a href="#">“CWAddProjectEntry” on page 104</a> <a href="#">“Adding a File to a Project” on page 226</a>

## **CWSetPluginOSSError**

Description	Informs the IDE of an OS error code generated during the current plug-in request.				
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWSetPluginOSSError(     <u>CWPluginContext</u> context,     <u>CWOSResult</u> result);</pre>				
Parameters	Parameters for this function are:  <table><tr><td>context</td><td>Private, opaque IDE state.</td></tr><tr><td>result</td><td>The error code generated by the host operating system.</td></tr></table>	context	Private, opaque IDE state.	result	The error code generated by the host operating system.
context	Private, opaque IDE state.				
result	The error code generated by the host operating system.				
Return	See <a href="#">“Result Codes for Plug-ins” on page 193.</a>				

Remarks	If a plug-in fails to act on a request because of an error returned by the host operating system, the plug-in may pass the error along to the IDE through <code>CWSetPluginOSError</code> .
See Also	<a href="#">“CWMacOSErrToCWResult” on page 136</a>

## CWShowStatus

Description	Presents progress information during a build operation.						
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWShowStatus(     <a href="#">CWPluginContext</a> context,     const char*     line1,     const char*     line2);</pre>						
Parameters	<p>Parameters for this function are:</p> <table> <tr> <td>context</td> <td>Private, opaque IDE state.</td> </tr> <tr> <td>line1,</td> <td>Pass text describing plug-in status in these arguments as C character strings.</td> </tr> <tr> <td>line2</td> <td></td> </tr> </table>	context	Private, opaque IDE state.	line1,	Pass text describing plug-in status in these arguments as C character strings.	line2	
context	Private, opaque IDE state.						
line1,	Pass text describing plug-in status in these arguments as C character strings.						
line2							
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .						
Remarks	Use <code>CWShowStatus</code> to display status information during a long operation. This call currently works only for linkers. The IDE displays <code>line1</code> and <code>line2</code> in the linker status window.						
See Also	<a href="#">“CWReportMessage” on page 140</a> <a href="#">“CWAlert” on page 105</a>						

## CWStorePluginData

Description	Associates arbitrary, persistently stored plug-in data with a file in the active target.
Prototype	<pre>#include "DropinCompilerLinker.h" CW_CALLBACK CWStorePluginData(     <a href="#">CWPluginContext</a> context,     long           whichfile,     <a href="#">CWDataType</a>      type,     <a href="#">CWMemHandle</a>   pluginData);</pre>
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
whichfile	Specifies the project file with which to associate data. To specify global data, that is data for the entire target and not a specific file, use <a href="#">kTargetGlobalPluginData</a> .
type	Specifies the kind of data. This is a four-character code assigned by the plug-in (usually having mnemonic value).
pluginData	A <a href="#">CWMemHandle</a> containing the data to be stored with the file.
Return	See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	CWStorePluginData stores data of type type contained in pluginData in the project and associates it with file whichfile. The format of this data is maintained and understood only by the plug-in, not the IDE. plug-ins may store any data they choose, and assign the data any meaningful 4-character type code.
<b>NOTE</b>	All 4-character type codes consisting entirely of lower case letters are reserved for use by Metrowerks.

---

Data stored with a file is public, in the sense that by requesting the same type of data for the same file, one plug-in can retrieve data stored by another. Although uncommon in practice, plug-ins should use distinctive type codes to reduce the probability of clashes.

whichfile specifies either [kTargetGlobalPluginData](#) to store data with the current target (rather than a specific file), or the number of a file in the active project target, based on the target's link order. File numbers range from 0 to the count returned by [CWGetProjectFileCount](#) - 1.

The data to be stored must reside in a [CWMemHandle](#) allocated using [CWAllocMemHandle](#). The handle becomes the property of the IDE.

Using different data types specified in the type argument, a plug-in may store more than kind of data for a file in the project. Calling

`CWStorePluginData` with a type of data that has already been stored will replace the original data.

Plug-in data associated with a file is kept in the project's target data file and thus persists across invocations of the plug-in and the IDE. Note that all data stored by `CWStorePluginData` is removed from the project when:

1. the user chooses **Project > Remove Object Code**
2. when the file is removed from the target
3. the user manually deletes the target data folder

See Also

[“CWGetPluginData” on page 129](#)

[“CWAllocMemHandle” on page 107](#)

## **CWUnlockMemHandle**

Description Tells the IDE that a plug-in is done dereferencing a [CWMemHandle](#).

Prototype 

```
#include "DropinCompilerLinker.h"
CW_CALLBACK CWUnlockMemHandle(
    CWPluginContext context,
    CWMemHandle handle);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.  
handle The handle to unlock.

Return See [“Result Codes for Plug-ins” on page 193](#).

Remarks After using a handle that has been locked, and when finished using any pointers to the data contained in the handle, plug-ins should call `CWUnlockMemHandle` to unlock the handle.

Do not use a pointer returned by [CWLockMemHandle](#) after unlocking its handle with `CWUnlockMemHandle`. Doing so can lead to serious and difficult to find memory bugs, since an unlocked handle's contents may be relocated (moved to a different memory address) at any time. Once a block moves, any existing pointers become invalid (“stale”) and writing to them will corrupt memory.

Calls to [CWLockMemHandle](#) and CWUnlockMemHandle are counted. A handle remains locked as long as its lock count is greater than zero. When initially allocated, handles are unlocked. [CWLockMemHandle](#) increases a handle's lock count, and CWUnlockMemHandle decreases its lock count.

See Also [“CWAllocMemHandle” on page 107](#)

[“CWFreeMemHandle” on page 112](#)

[“CWGetMemHandleSize” on page 124](#)

[“CWResizeMemHandle” on page 141](#)

[“CWLockMemHandle” on page 134](#)

## **CWUserBreak**

Description Checks to see if the user has cancelled the current operation.

Prototype  
`#include "DropinCompilerLinker.h"  
CW_CALLBACK CWUserBreak(  
    CWPluginContext context);`

Parameters Parameters for this function are:

`context` Private, opaque IDE state.

Return See [“Result Codes for Plug-ins” on page 193](#).

Remarks Call this routine regularly while acting on a request that requires more than several seconds of processing time.

If CWUserBreak returns [cwErrUserCanceled](#), the plug-in should stop processing, clean up, and return to the IDE as soon as possible.

See Also [“CWReportMessage” on page 140](#)

# User Routines for Plug-ins

This section describes entry points common to all plug-ins:

- [Main Plug-in Entry Point](#)
- [CWPlugin\\_GetDropInFlags Entry Point](#)

- [CWPlugin\\_GetDropInName Entry Point](#)
- [CWPlugin\\_GetDisplayName Entry Point](#)
- [CWPlugin\\_GetPluginInfo](#)

## Main Plug-in Entry Point

Description Provides the main plug-in entry point, for servicing functional requests from the IDE.

Prototype `CWPlugin_ENTRY (main) (CWPluginContext context)`

Parameters Parameters for this entry point are:

`CWPluginContext` Private, opaque IDE state.

Return See [“Result Codes for Plug-ins” on page 193](#).

Remarks This entry point is used by the IDE to request services from the plug-in. All plug-ins must implement this function.

To determine what service is being requested by the IDE, plug-ins should call [CWGetPluginRequest](#), which returns a value indicating the service to perform. Not all plug-ins receive or need to respond to all service requests.

Prior to returning from its main entry point, a plug-in should call [CWDonePluginRequest](#) with an error code of type [CWResult](#), which should be `cwNoErr`, if no error occurred, or `cwErrRequestFailed`, if the request was not handled. [CWDonePluginRequest](#) then returns an error code to the plug-in, which the plug-in should return to the IDE, as its `main` function result.

The main entry point is passed a single parameter of type [CWPluginContext](#) which maintains state private to the IDE, on behalf of the plug-in. This state assists the IDE in providing services to the plug-in, and must be passed to the IDE in all calls made by the plug-in.

See Also [“CWGetPluginRequest” on page 130](#)

[“CWDonePluginRequest” on page 109](#)

[“CWPluginContext” on page 172](#)

[“reqInitialize” on page 192](#)

[“reqTerminate” on page 193](#)

## **CWPlugin\_GetDropInFlags Entry Point**

Description	Provides an informational entry point to the IDE that the IDE calls to learn about plug-in capabilities.				
Prototype	<pre>#include &lt;CWPlugins.h&gt; CWPlugin_ENTRY (CWPlugin_GetDropInFlags)     (const DropInFlags**, long* flagsSize);</pre>				
Parameters	Parameters for this entry point are:				
	<table><tr><td>flags</td><td>Returns a <a href="#">DropInFlags</a> structure to the IDE specifying information about a plug-in and its capabilities.</td></tr><tr><td>flagsSize</td><td>Specifies the size of the structure returned in <code>flags</code>.</td></tr></table>	flags	Returns a <a href="#">DropInFlags</a> structure to the IDE specifying information about a plug-in and its capabilities.	flagsSize	Specifies the size of the structure returned in <code>flags</code> .
flags	Returns a <a href="#">DropInFlags</a> structure to the IDE specifying information about a plug-in and its capabilities.				
flagsSize	Specifies the size of the structure returned in <code>flags</code> .				
Return	Usually, the plug-in should return the <a href="#">cwNoErr</a> result code. See <a href="#">“Result Codes for Plug-ins” on page 193</a> .				
Remarks	This entry point returns information about a plug-in to the IDE, by filling in a <code>DropInFlags</code> structure. Usually, this is implemented by returning a pointer to a constant <code>DropInFlags</code> structure.  The <code>flagsSize</code> parameter should be set to the size of the structure being returned (normally obtained by applying the C <code>sizeof</code> operator to the <code>DropInFlags</code> type).				
<b>NOTE</b>	COM-only plug-ins do not need to specify flags. All other plug-ins are required to do so.				
Mac OS	On Mac OS, this routine is optional. In its place, a plug-in may provide a <a href="#">‘Flag’ Resource</a> , specifying the same information.				
See Also	<a href="#">“DropInFlags” on page 182</a> <a href="#">“Specifying Plug-in Capabilities” on page 65</a> <a href="#">“Compiler and Linker DropIn Flags” on page 199</a>				

## **CWPlugin\_GetDropInName Entry Point**

Description	Provides an informational entry point that the IDE uses to determine the plug-in's dropin name.
Prototype	#include <CWPlugins.h> CWPlugin_ENTRY (CWPlugin_GetDropInName) (const char** dropinName)
Parameters	Parameters for this entry point are:
	dropinName                         Returns a C string containing the drop-in name of the plug-in to the IDE.
Return	Usually, the plug-in should return the <code>cwNoErr</code> result code. See <a href="#">“Result Codes for Plug-ins” on page 193</a> .
Remarks	This entry point is used by the IDE to determine the plug-in's dropin name, which is used to keep track of plug-ins in the project file. All plug-ins must implement this entry point.
<b>NOTE</b>	The CWPlugin_GetDropInName entry is deprecated in favor of the new <a href="#">CWPlugin_GetPluginInfo</a> entry. GetDropInName will be called if the dropin name is not set by <a href="#">CWPlugin_GetPluginInfo</a> .
See Also	<a href="#">“Result Codes for Plug-ins” on page 193</a>

## **CWPlugin\_GetDisplayName Entry Point**

Description	Provides an informational entry point that the IDE uses to determine the plug-in's display name.
Prototype	#include <CWPlugins.h> CWPlugin_ENTRY (CWPlugin_GetDisplayName) (const char** displayName);
Parameters	Parameters for this entry point are:
	displayName                         Returns a C string containing the display name of the plug-in to the IDE.
Return	Usually, the plug-in should return the <code>cwNoErr</code> result code. See <a href="#">“Result Codes for Plug-ins” on page 193</a> .

## Plug-in API Reference

### CWPlugin\_GetPluginInfo

---

Remarks	This entry point is used by the IDE to determine the plug-in's display name, which is used when displaying the name of the plug-in to the user (when selecting a compiler or linker in <b>Target Settings</b> , for example).
<b>NOTE</b>	The CWPlugin_GetDisplayName entry is deprecated in favor of the new <a href="#">CWPlugin_GetPluginInfo</a> entry. GetDropInName will be called if the dropin name is not set by <a href="#">CWPlugin_GetPluginInfo</a> .
Mac OS	On Mac OS, this routine is optional. In its place, a plug-in may provide a <a href="#">"STR ' Resource"</a> , specifying the same information.
See Also	<a href="#">"Result Codes for Plug-ins" on page 193</a>

## CWPlugin\_GetPluginInfo

Description	Provides an informational entry point that the IDE uses to determine the plug-in's display name.		
Prototype	<pre>#include &lt;CWPlugins.h&gt; CWPLUGIN_ENTRY (CWPlugin_GetPluginInfo) (const CWPluginInfo** pluginInfo);</pre>		
Parameters	Parameters for this entry point are:		
	<table><tr><td>CWPluginInfo</td><td>A <a href="#">CWPluginInfo</a> object containing information about the plug-in.</td></tr></table>	CWPluginInfo	A <a href="#">CWPluginInfo</a> object containing information about the plug-in.
CWPluginInfo	A <a href="#">CWPluginInfo</a> object containing information about the plug-in.		
Return	Usually, the plug-in should return the <code>cwNoErr</code> result code. See <a href="#">"Result Codes for Plug-ins" on page 193</a> .		
Remarks	Use this function to get information about the plug-in, including the dropin name and the display name.		
<b>NOTE</b>	The compiler checks for information from the CWPlugin_GetPluginInfo function. Then it looks for information from the <a href="#">CWPlugin_GetDropInName Entry Point</a> and <a href="#">CWPlugin_GetDisplayName Entry Point</a> functions. Finally, it looks for a resource file. For this reason, using CWPlugin_GetPluginInfo is more efficient than the other two ways of providing this information.		

The CWPlugin\_GetPluginInfo function supercedes the [CWPlugin\\_GetDropInName Entry Point](#) and [CWPlugin\\_GetDisplayName Entry Point](#) functions.

Mac OS On Mac OS, this routine is optional. In its place, a plug-in may provide a ["STR ' Resource"](#), specifying the same information.

See Also ["Result Codes for Plug-ins" on page 193](#)

## Data Structures for Plug-ins

This section describes the data structures and data types available to all plug-ins.

The plug-in API's data structures are:

- [CWAcessPathInfo](#)
- [CWAcessPathListInfo](#)
- [CWAcessPathType](#)
- [CWAddr64](#)
- [CWDataType](#)
- [CWDependencyType](#)
- [CWFileInfo](#)
- [CWFfileName](#)
- [CWFfileSpec](#)
- [CWFfileTime](#)
- [CWFrameworkInfo](#)
- [CWIDEInfo](#)
- [CWMemHandle](#)
- [CWMessagRef](#)
- [CWNewProjectEntryInfo](#)
- [CWNewTextDocumentInfo](#)
- [CWOSResult](#)
- [CWOverlay1FileInfo](#)
- [CWOverlay1GroupInfo](#)
- [CWOverlay1Info](#)

- [CWPluginContext](#)
- [CWPluginInfo](#)
- [CWProjectFileInfo](#)
- [CWProjectSegmentInfo](#)
- [CWRelativePath](#)
- [CWRelativePathFormat](#)
- [CWRelativePathTypes](#)
- [CWResult](#)
- [DropInFlags](#)

## CWAcessPathInfo

Description	Returns information about a single access path.	
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef struct CWAcessPathInfo {     CWFileSpec pathSpec;     Boolean    recursive;     long       subdirectoryCount; } CWAcessPathInfo;</pre>	
Fields	The fields in CWAcessPathInfo are:	
	pathSpec	Specifies a project path (directory).
	recursive	Indicates whether the IDE examines subdirectories of pathSpec to find project files. If true, the IDE searches subdirectories.
	subdirectoryCount	Returns the count of subdirectories in pathSpec. This field will be zero if recursive is false.
Remarks	<p><a href="#">CWGetAccessPathInfo</a> returns this structure to describe a file access path in the current project. pathSpec indicates the location of the access path. If recursive is true, the plug-in may obtain each of the subdirectoryCount subdirectories by calling <a href="#">CWGetAccessPathSubdirectory</a> with an index ranging from 0 to subdirectoryCount - 1.</p>	

See Also [“CWGetAccessPathInfo” on page 113](#)

[“CWGetAccessPathSubdirectory” on page 115](#)

## CWAcessPathListInfo

Description Returns properties and counts of the access paths in the current target.

Definition

```
#include <CWPlugins.h>
typedef struct CWAcessPathListInfo
{
    long      systemPathCount;
    long      userPathCount;
    Boolean   alwaysSearchUserPaths;
    Boolean   convertPaths;
} CWAcessPathListInfo;
```

Fields The fields in CWAcessPathListInfo are:

systemPathCount Returns the number of system paths specified in **Edit > Target Settings**, in the **Access Paths** panel.

userPathCount Returns the number of user paths specified in **Edit > Target Settings**, in the **Access Paths** panel.

alwaysSearchUserPaths Returns true if the IDE has been configured to always search for files in the user access paths (regardless of #include syntax).

convertPaths Indicates that a Mac OS hosted IDE will attempt to interpret special characters appearing in file specifications, such as ‘\’ and ‘/’, as if they were DOS and Unix path separators. Not used on other host platforms.

Remarks [CWGetAccessPathListInfo](#) returns a CWAcessPathListInfo structure containing information about the access paths specified for the current target. systemPathCount and userPathCount indicate the total number of system and user

---

access paths. plug-ins typically call [CWGetAccessPathListInfo](#) to determine the number of user and system paths, prior to enumerating them using [CWGetAccessPathInfo](#).

`alwaysSearchUserPaths` indicates whether the IDE searches user paths when looking for “system includes.” When set, system include statements of the form:

```
#include <headerfile.h>
```

search the user paths first, as well as system paths, just as if they were written as user includes:

```
#include "headerfile.h"
```

When `alwaysSearchUserPaths` is true, plug-ins interfacing with command line tools should include all user search paths on the command line as system search paths.

Mac OS      `convertPaths` indicates whether a Mac OS hosted IDE will attempt to interpret file specifications, such as include file paths, as if they were DOS or Unix paths.

See Also     [“CWGetAccessPathListInfo” on page 114](#)

[“CWGetAccessPathInfo” on page 113](#)

## CWAcessPathType

Description    Specifies a kind of access path (user or system).

Definition    

```
#include <CWPlugins.h>
typedef enum CWAcessPathType
{
    cwSystemPath,
    cwUserPath
} CWAcessPathType;
```

Fields       The members of CWAcessPathType are:

    cwSystemPath    Specifies a system access path.

    cwUserPath      Specifies a user access path.

Remarks	Values of type <code>CWAccessPathType</code> are used to specify user or system paths when calling <a href="#">CWGetAccessPathInfo</a> and <a href="#">CWGetAccessPathListInfo</a> .
See Also	<a href="#">“CWGetAccessPathInfo” on page 113</a> <a href="#">“CWGetAccessPathListInfo” on page 114</a>

## CWAddr64

Description	Stores a 64-bit address.				
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef struct CWAddr64 {     long    lo;     long    hi; } CWAddr64;</pre>				
Fields	The fields in <code>CWAddr64</code> are: <table> <tr> <td><code>lo</code></td><td>Holds the lower 32 bits of a 64-bit address.</td></tr> <tr> <td><code>hi</code></td><td>Holds the upper 32 bits of a 64-bit address.</td></tr> </table>	<code>lo</code>	Holds the lower 32 bits of a 64-bit address.	<code>hi</code>	Holds the upper 32 bits of a 64-bit address.
<code>lo</code>	Holds the lower 32 bits of a 64-bit address.				
<code>hi</code>	Holds the upper 32 bits of a 64-bit address.				
Remarks	Use the <code>CWAddr64</code> data structure to hold a 64-bit memory address. This data structure is used in the <a href="#">CWOverlay1GroupInfo</a> data structure.				
See Also	<a href="#">“CWOverlay1GroupInfo” on page 171</a>				

## CWDataType

Description	Uniquely identifies a category or type of data. Also used for specifying unique identifiers.
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef unsigned long CWDataType;</pre>
Remarks	Used to identify the type of custom data associated by a plug-in with a project file or a target as a whole. Must be a four-character constant.
	This type is also used to hold unique, mnemonic 4-character identifiers or descriptors. For example, CPU and OS codes returned

by a [CWPlugin GetTargetList Entry Point](#) are unique 4-character codes of type `CWDataType`.

---

**NOTE** All lower-case 4-character constants are reserved for use by Metrowerks.

---

See Also [“CWStorePluginData” on page 145](#)

[“CWGetPluginData” on page 129](#)

## **CWDependencyType**

Description Specifies how one source file relies on the contents of another.

Definition

```
#include <CWPlugins.h>
typedef enum CWDependencyType {
    cwNoDependency,
    cwNormalDependency,
    cwInterfaceDependency
} CWDependencyType;
```

Remarks This data type is used in [CWFFileInfo](#) to indicate the type of source dependency to establish between files.

`cwNoDependency` specifies that there is no dependency to establish. This useful when using [CWFFindAndLoadFile](#) to simply load a file, rather than establish a dependency.

`cwNormalDependency` specifies that a dependent file depends upon the entire contents of the file. Whenever the depended-upon file is modified, the dependent file will be marked dirty and recompiled. A C/C++ compiler uses this type of dependency to specify dependencies on header files.

`cwInterfaceDependency` specifies that the dependent file depends on the external interface of the file. With this type of dependency, the dependent files are not marked dirty when the depended-upon file is modified. They are only marked dirty when the depended-upon file is compiled and the compiler indicates that the interface has changed by setting the `interfaceChanged` flag in the [CWObjectData](#) data structure for the [CWStoreObjectData](#)

call to the IDE. It's up to the plug-in to determine what is exported through the external interface and when it has changed.

For `cwInterfaceDependency`, the plug-in typically will generate the data describing the external interface and save it using [`CWStorePluginData`](#). On subsequent compiles it recomputes the interface data, retrieves the interface data from the last time it was compiled using [`CWGetPluginData`](#) and checks to see if the interface data has changed. If it has, it stores the new interface data and sets the `interfaceChanged` flag when it calls [`CWStoreObjectData`](#).

See Also

["CWFileInfo" on page 159](#)

["CWFindAndLoadFile" on page 110](#)

["CWStorePluginData" on page 145](#)

["CWGetPluginData" on page 129](#)

["CWStoreObjectData" on page 267](#)

## CWFileInfo

Description      Holds information returned by [`CWFindAndLoadFile`](#).

Definition

```
#include <CWPlugins.h>
typedef struct CWFileInfo {
    Boolean      fullsearch;
    char         dependencyType;
    long         isdependentoffile;
    Boolean      suppressload;
    Boolean      padding;
    const char*  filedatal;
    long         filedatalength;
    short        filedatatype;
    short        fileID;
    CWfileSpec   filespec;
    Boolean      alreadyincluded;
    Boolean      recordbrowseinfo;
} CWFileInfo;
```

Fields      The fields in `CWFileInfo` are:

fullsearch	Specify true to tell the IDE to search the user access paths and then the system paths for the file. Specify false to tell the IDE to search the system paths only.
dependencyType	Set this field to a value from <a href="#">CWDependencyType</a> to specify how the file to be searched for is depended upon by the file specified in the <code>isdependentoffile</code> field.
isdependentoffile	Specifies the project file (by index) that depends on the file to search for (specified in the <code>filename</code> parameter in a call to <a href="#">CWFindAndLoadFile</a> ). Set this to <a href="#">kCurrentCompiledFile</a> to specify the file that the plug-in is currently processing.
suppressload	Specify true to tell the IDE to find a file without loading it into memory. If set to false, the IDE will load the file into memory if it's found and it's a text file or cached precompiled header file.
padding	Reserved for use by the IDE. Do not modify the contents of this field.
filedata	Returns a pointer to the file data, if the IDE found the file. The IDE returns NULL if the file was not found or the file was not loaded into memory. If the file was found but not loaded into memory, the IDE stores a specification for the file in the <code>filespec</code> field.
filedatalength	Returns the size of the file data, in bytes, if the file was loaded into memory. Returns 0 if the file was not loaded into memory.

filedatatype	Returns the kind of data in the file, if found. Possible values for this field are <a href="#">cwFileTypePrecompiledHeader</a> , <a href="#">cwFileTypeText</a> , or <a href="#">cwFileTypeUnknown</a>
fileID	Returns a unique number for the file if found. This field is only valid when used by compiler plug-ins.
filespec	Returns the file specification of the file being searched for. The IDE stores a file specification in this field whether or not it loads the file's contents into memory. This field is only valid when used by compiler plug-ins.
alreadyincluded	Returns true if the file has already been searched for in the current operation. This field is only valid when used by compiler plug-ins.
recordbrowseinfo	Returns true to tell the plug-in to generate browser data for the file. This field is only valid when used by compiler plug-ins.

Remarks A plug-in fills in fields in this structure to specify how the IDE should search for a file through [CWFindAndLoadFile](#).

See Also [“CWDependencyType” on page 158](#)

[“CWFindAndLoadFile” on page 110](#)

[“cwFileTypePrecompiledHeader” on page 187](#)

[“cwFileTypeText” on page 187](#)

[“cwFileTypeUnknown” on page 188](#)

[“kCurrentCompiledFile” on page 189](#)

## CWFileName

Description Specifies the name of a file.

Definition

```
#include <CWPlugins.h>
#if CWPlugin_API == CWPlugin_API_MACOS
    typedef char          CWFileName[32];
#elif CWPlugin_API == CWPlugin_API_WIN32
    typedef char          CWFileName[65];
#endif
```

Remarks Use this type to specify the name of a file. For example, [CWProjectFileInfo](#) uses this type to specify a file name.

---

**NOTE** The definition of this type is platform dependent.

---

See Also [“CWProjectFileInfo” on page 174](#)

## CWFileSpec

Description Represents the data type used by the host operating system to specify a file.

Definition

```
#include <CWPlugins.h>
#if CWPLUGIN_API == CWPLUGIN_API_MACOS

    #if CWPLUGIN_LONG_FILENAME_SUPPORT

        #define CWPLUGIN_FILENAME_LEN 256
        typedef struct CWFileSpec
        {
            FSRef           parentDirRef; /* parent
directory */
            HFSUniStr255 filename;      /* unicode file
name */
        } CWFileSpec;

        typedef char
CWFileName[CWPLUGIN_FILENAME_LEN];

    #else

        #define CWPLUGIN_FILENAME_LEN 32
    
```

```
typedef FSSpec CWFileSpec;
typedef char
CWFileName[CWPLUGIN_FILENAME_LEN];

#endif

typedef unsigned long CWFleTime;
typedef OSerr          CWOSResult;

#ifelif CWPLUGIN_API == CWPLUGIN_API_WIN32

typedef unsigned char Boolean;
typedef struct CWfileSpec { char
path[MAX_PATH]; } CWfileSpec;

#if CWPLUGIN_LONG_FILENAME_SUPPORT

#define CWPLUGIN_FILENAME_LEN 256
typedef char
CWFileName[CWPLUGIN_FILENAME_LEN];

#else

#define CWPLUGIN_FILENAME_LEN 65
typedef char
CWFileName[CWPLUGIN_FILENAME_LEN];

#endif

typedef FILETIME CWFleTime;
typedef DWORD      CWOSResult;

#ifelif CWPLUGIN_API == CWPLUGIN_API_UNIX

#define MAX_PATH           MAXPATHLEN
#define CWPLUGIN_FILENAME_LEN 65
#ifndef __MACTYPES__
typedef unsigned char Boolean;
#endif

#ifndef FALSE
#define FALSE 0
```

```
#endif

#ifndef TRUE
#define TRUE 1
#endif

typedef struct CWFileSpec { char
path[MAX_PATH]; } CWFileSpec;
typedef char
CWFileName[CWPLUGIN_FILENAME_LEN];
typedef time_t           CWFileType;
typedef int               CWOSResult;

#endif
```

**Remarks** The plug-in API defines this data type to be equivalent to the data type used by the host operating system to specify a file. On Windows, this is a full path. On Mac OS, this is a `FileSpec`.

---

**NOTE** The definition of this type is platform dependent.

---

**See Also** [“CWFileInfo” on page 159](#)

[“CWGetProjectFile” on page 131](#)

## **CWFileType**

**Description** Specifies the modification date and time of a file.

**Definition**

```
#include <CWPlugins.h>
#if CWPlugin_API == CWPlugin_API_MACOS
    typedef unsigned long   CWFileType;
#elif CWPlugin_API == CWPlugin_API_WIN32
    typedef FILETIME        CWFileType;
#endif
```

**Remarks** The plug-in API defines this data type to be equivalent to the data type used by the host operating system to represent a date and time value.

---

**NOTE** The definition of this type is platform dependent.

---

See Also [“CWProjectFileInfo” on page 174](#)

[“CWSetModDate” on page 143](#)

## CWFrameworkInfo

Description Holds information about a framework.

Definition

```
#include <DropInCompilerLinker.h>
typedef struct CWFrameworkInfo {
    CWFFileSpec fileSpec;
    char         version[256];
} CWFrameworkInfo;
```

Fields The fields in CWIDEInfo are:

fileSpec	Location of ".framework" directory
version[256]	Which version directory to use

Remarks If the version string is empty, the IDE uses the “Current” symbolic link.

See Also [“CWGetFrameworkInfo” on page 123](#)

## CWIDEInfo

Description Holds information about the IDE.

Definition

```
#include <CWPlugins.h>
typedef struct CWIDEInfo
{
    unsigned short    majorVersion;
    unsigned short    minorVersion;
    unsigned short    bugFixVersion;
    unsigned short    buildVersion;
    unsigned short    dropinAPIVersion;
} CWIDEInfo;
```

Fields The fields in CWIDEInfo are:

majorVersion	Major IDE version number.
minorVersion	Minor IDE version number.
bugFixVersion	Revision number for IDE.

	buildVersion	Build number of IDE.
	dropinAPIVersion	Latest plug-in API version available with this IDE.
Remarks	The IDE returns this structure in response to a <a href="#">CWGetIDEInfo</a> call. It provides detailed IDE version information, and the plug-in API version.	
<b>NOTE</b>	The API version returned in this structure is always the latest available, rather than the API version the IDE is using to run the current plug-in, as reported by <a href="#">CWGetAPIVersion</a> .	
See Also	<a href="#">“CWGetIDEInfo” on page 124</a> <a href="#">“CWGetAPIVersion” on page 116</a>	

## CWMemHandle

Description	Maintains a resizable memory block by indirect reference.
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef struct     CWMemHandlePrivateStruct* CWMemHandle;</pre>
Remarks	Variables of type <code>CWMemHandle</code> store blocks of memory allocated, freed, and referenced through plug-in API routines. <code>CWMemHandle</code> blocks, often referred to just as handles, are general purpose memory blocks, much like pointer blocks.  However, plug-ins cannot access the contents of <code>CWMemHandles</code> directly, as with pointers. Instead, plug-ins must call <a href="#">CWLockMemHandle</a> to obtain a pointer to a handle's contents. When finished accessing a handle, a plug-in should call <a href="#">CWUnlockMemHandle</a> .  Handles can be allocated using <a href="#">CWAllocMemHandle</a> , freed using <a href="#">CWFreeMemHandle</a> , and resized using <a href="#">CWResizeMemHandle</a> . The current size of a handle can be determined using <a href="#">CWGetMemHandleSize</a> .
See Also	<a href="#">“CWAllocMemHandle” on page 107</a>

[“CWFreeMemHandle” on page 112](#)

[“CWResizeMemHandle” on page 141](#)

[“CWGetMemHandleSize” on page 124](#)

[“CWLokMemHandle” on page 134](#)

[“CWUnlockMemHandle” on page 147](#)

[“Handles” on page 82](#)

## CWMessageRef

Description      Specifies the location of the source code that generated a compiler error or warning.

Definition     

```
#include <CWPlugins.h>
typedef struct CWMessageRef
{
    CWFileSpec      sourcefile;
    long             linenumber;
    short            tokenoffset;
    short            tokenlength;
    long             selectionoffset;
    long             selectionlength;
} CWMessageRef;
```

Fields        The fields in CWMessageRef are:

sourcefile      Specifies the source file that caused the error or warning.

linenumber      Specifies the line of text where the error or warning occurred.

tokenoffset     Specifies the beginning of the item within the line of source code that caused the error or warning.

tokenlength     Specifies the length of the offending token. The IDE underlines the token when displayed.

	<code>selectionoffset</code>	Specifies the beginning of the text to highlight, when the user selects an error or warning item in the message window. The offset is specified relative to the start of the source file.
	<code>selectionlength</code>	Specifies the length of the text to select, when the user selects an error or warning item in the message window.
Remarks	A <code>CWMessageRef</code> structure describes the source file location at which an error or warning occurred. It provides sufficient information for the IDE to locate and display the source code associated with a message. A plug-in passes this structure to the IDE in a call to <a href="#">CWReportMessage</a> .	
See Also	<a href="#">“CWReportMessage” on page 140</a>	

## **CWNewProjectEntryInfo**

Description	Specifies placement information and file flags for a file added to a project.
Definition	<pre>typedef struct CWNewProjectEntryInfo {     long    position;     long    segment;     long    overlayGroup;     long    overlay;     const char *groupPath;     Boolean mergeintooutput;     Boolean weakimport;     Boolean initbefore; } CWNewProjectEntryInfo;</pre>
Fields:	The fields in <code>CWNewProjectEntryInfo</code> are:

<code>position</code>	Specifies the link order position.
<code>segment</code>	Specifies the segment number.
<code>overlayGroup</code>	Specifies the overlay group number.
<code>overlay</code>	Specifies the overlay group number.

groupPath      Specifies the project group to which the file should be added. This is a C-style string specifying the full path to a group, using a colon (:) to delimit the groups. For example, "Group1:Group2" is a valid string. If the group specified is not found, it will be created.

mergeintooutput      Specifies the value for the the "Merge Into Output" setting. Used only in Mac OS projects.

weakimport      Specifies the value for the the "Import Weak" setting. Used only in Mac OS projects.

initbefore      Specifies the value for the the "Initialize Before" setting. Used only in Mac OS projects.

Remarks      This structure specifies where a file added to a project using [CWAddProjectEntry](#) will appear in the various project file orderings (link, overlay, and segment). In addition, it provides values for several per-file target settings.

Some of the indexes provided are optional, and will be ignored by the IDE if not relevant. For example, Windows projects will not have segments or overlays, and thus the segment, overlayGroup, and overlay fields will be ignored.

Plug-ins may specify [kDefaultLinkPosition](#) for any of the position, segment, overlayGroup, and overlay fields. This tells the IDE to add the file in the default position in the corresponding project ordering.

Each of the mergeintooutput, weakimport, and initbefore flags is ignored if the specified file already exists in a project. Also, the flags are ignored if they aren't relevant to the file being added to the project. These flags are only used by projects containing Mac OS targets (which can exist on any platform).

See Also      [“CWAddProjectEntry” on page 104](#)

## **CWNewTextDocumentInfo**

Description	Specifies information about a newly created editor window						
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef struct CWNewTextDocumentInfo {     const char*    documentname;     <a href="#">CWMemHandle</a>   text;     Boolean        markDirty; } CWNewTextDocumentInfo;</pre>						
Fields	The fields in <code>CWNewTextDocumentInfo</code> are:						
	<table><tr><td><code>documentname</code></td><td>Specifies the title of the document, as a C string.</td></tr><tr><td><code>text</code></td><td>Provides the text handle for the document.</td></tr><tr><td><code>markDirty</code></td><td>Specifies whether the IDE should regard the document as saved. Specify true to indicate that the document should be saved prior to closing, and false to discard the document's content upon closing.</td></tr></table>	<code>documentname</code>	Specifies the title of the document, as a C string.	<code>text</code>	Provides the text handle for the document.	<code>markDirty</code>	Specifies whether the IDE should regard the document as saved. Specify true to indicate that the document should be saved prior to closing, and false to discard the document's content upon closing.
<code>documentname</code>	Specifies the title of the document, as a C string.						
<code>text</code>	Provides the text handle for the document.						
<code>markDirty</code>	Specifies whether the IDE should regard the document as saved. Specify true to indicate that the document should be saved prior to closing, and false to discard the document's content upon closing.						
Remarks	The plug-in uses this structure to specify the title and contents of an editor window created with <a href="#">CWCreateNewTextDocument</a> . The <code>text</code> handle becomes the IDE's property; plug-ins should not dispose it.						
See Also	<a href="#">“CWCreateNewTextDocument” on page 108</a>						

## **CWOSResult**

Description	Represents the data type used by the host operating system to specify an error code.
Definition	<pre>#include &lt;CWPlugins.h&gt;</pre>
Remarks	The plug-in API defines this data type to be equivalent to the data type used by the host operating system to specify an error condition.
NOTE	The definition of this type is platform dependent.

See Also [“CWGetCallbackOSError” on page 117](#)

[“CWSetPluginOSError” on page 144](#)

## CWOverlay1FileInfo

Description Describes a file in an overlay.

Definition 

```
#include <CWPlugins.h>
typedef struct CWOverlay1FileInfo {
    long          whichfile;
} CWOverlay1FileInfo;
```

Fields The fields in CWOverlay1FileInfo are:

whichfile Returns the index of the overlay file in the file list view of the current target.

Remarks Use [CWGetOverlay1FileInfo](#) to iterate through files. Use whichfile in calls taking a file number, such as [CWLoadObjectData](#) or [CWGetFileInfo](#).

See Also [“CWGetOverlay1FileInfo” on page 126](#)

[“CWLoadObjectData” on page 263](#)

[“CWGetFileInfo” on page 120](#)

## CWOverlay1GroupInfo

Description Describes an overlay group.

Definition 

```
#include <CWPlugins.h>
typedef struct CWOverlay1GroupInfo {
    char          name[256];
    CWAddr64      address;
    long          numoverlays;
} CWOverlay1GroupInfo;
```

Fields The fields in CWOverlay1GroupInfo are:

name Returns the name of the overlay, as a C string.

	<b>address</b>	Returns the 64-bit absolute load address of the overlay.
	<b>numoverlays</b>	Returns the number of overlays in this group.
<b>Remarks</b>	Use <a href="#">CWGetOverlay1GroupInfo</a> to iterate through overlay groups.	
<b>See Also</b>	<a href="#">“CWGetOverlay1GroupInfo” on page 127</a>	

## **CWOverlay1Info**

<b>Description</b>	Returns information about one overlay.				
<b>Definition</b>	#include <CWPlugins.h> typedef struct CWOverlay1Info { char name[256]; long numfiles; } CWOverlay1Info;				
<b>Fields</b>	The fields in CWOverlay1Info are:  <table><tr><td><b>name</b></td><td>Returns the name of the overlay, as a C string.</td></tr><tr><td><b>numfiles</b></td><td>Returns the number of files compiled into this overlay.</td></tr></table>	<b>name</b>	Returns the name of the overlay, as a C string.	<b>numfiles</b>	Returns the number of files compiled into this overlay.
<b>name</b>	Returns the name of the overlay, as a C string.				
<b>numfiles</b>	Returns the number of files compiled into this overlay.				
<b>Remarks</b>	Use <a href="#">CWGetOverlay1Info</a> to iterate through overlay groups.				
<b>See Also</b>	<a href="#">“CWGetOverlay1Info” on page 129</a>				

## **CWPluginContext**

<b>Description</b>	Opaque reference to IDE state.
<b>Definition</b>	#include <CWPlugins.h> typedef struct CWPluginPrivateContext* CWPluginContext;
<b>Remarks</b>	This data type maintains information for the IDE, during calls to a plug-in. This information is private to the IDE, and should never be used directly by plug-ins.
	The <a href="#">Main Plug-in Entry Point</a> for a plug-in receives a parameter of this type, which must be preserved for later use and passed back to the IDE in the context parameter of any services the plug-in uses.

The informational plug-in entry points do not receive such a parameter, because they simply report static information back to the IDE, and should not require IDE services.

For more information, see [“Main Entry Point context Parameter” on page 77](#).

## CWPluginInfo

Description	Contains information about a plug-in.														
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef struct CWPluginInfo {     short           version;     const char*    companyName;     const char*    pluginName;     const char*    pluginDisplayName;     const char*    familyName;     unsigned short majorIDEVersion; // Required     unsigned short minorIDEVersion; } CWPluginInfo;</pre>														
Fields	<p>The fields in CWPluginInfo are:</p> <table><tr><td>version</td><td>Optional.</td></tr><tr><td>companyName</td><td>Optional. For example, Metrowerks.</td></tr><tr><td>pluginName</td><td>Optional. The name of your plug-in module. If you don't provide this name, the compiler uses Dropin-&gt;GetName(). Otherwise, it looks for a resource file.</td></tr><tr><td>pluginDisplayName</td><td>The name of your plug-in, as you would like users to see it in the CodeWarrior™ IDE.</td></tr><tr><td>familyName</td><td>For example, “Java”.</td></tr><tr><td>majorIDEVersion</td><td>This field is required.</td></tr><tr><td>minorIDEVersion</td><td>Optional.</td></tr></table>	version	Optional.	companyName	Optional. For example, Metrowerks.	pluginName	Optional. The name of your plug-in module. If you don't provide this name, the compiler uses Dropin->GetName(). Otherwise, it looks for a resource file.	pluginDisplayName	The name of your plug-in, as you would like users to see it in the CodeWarrior™ IDE.	familyName	For example, “Java”.	majorIDEVersion	This field is required.	minorIDEVersion	Optional.
version	Optional.														
companyName	Optional. For example, Metrowerks.														
pluginName	Optional. The name of your plug-in module. If you don't provide this name, the compiler uses Dropin->GetName(). Otherwise, it looks for a resource file.														
pluginDisplayName	The name of your plug-in, as you would like users to see it in the CodeWarrior™ IDE.														
familyName	For example, “Java”.														
majorIDEVersion	This field is required.														
minorIDEVersion	Optional.														

**Remarks** CWPluginInfo is passed by CWPlugin\_GetPluginInfo entry point to provide the IDE with plug-in identity info.

---

**NOTE** You can get better speed from the compiler by providing a value in the pluginName field.

---

## **CWProjectFileInfo**

**Description** Contains information about a file in a project.

**Definition**

```
#include <CWPlugins.h>
typedef struct CWProjectFileInfo
{
    CWFfileSpec      filespec;
    CWFfileTime     moddate;
    short              segment;
    Boolean            hasobjectcode;
    Boolean            hasresources;
    Boolean            isresourcefile;
    Boolean            weakimport;
    Boolean            initbefore;
    Boolean            gendebbug;
    CWFfileTime     objmoddate;
    CWFfileName    dropinname;
    short              fileID;
    Boolean            recordbrowseinfo;
    Boolean            reserved;

#if CWPlugin_HOST == CWPlugin_HOST_MACOS
    OSType             filetype;
    OSType             filecreator;
#endif
    Boolean            hasunitdata;
    Boolean            mergeintooutput;
    unsigned long      unitdatadependencytag;
} CWProjectFileInfo;
```

**Fields** The fields in CWProjectFileInfo are:

filespec	Returns the file specification for the file.
moddate	Returns the date and time that the file was last modified.
segment	Returns the number of the segment that the file will be linked into.
hasobjectcode	Indicates whether or not the file generates object code (not whether any actually exists).
hasresources	Indicates whether or not the file generates resource data (not whether any actually exists).
isresourcefile	Indicates whether or not the file is a binary resource file that will be linked into the final executable.
weakimport	Indicates whether or not the file's <b>Import Weak</b> flag is set.
initbefore	Indicates whether or not the file's <b>Init Before</b> flag is set.
gendebug	Indicates whether or not debugging information should be generated for the file.
objmoddate	Indicates the date and time at which the file's object code was last modified.
dropinname	Returns the name of the plug-in that the IDE calls on to process the file.
fileID	Returns a unique number, used in browse records to identify source files.

---

	recordbrowseinfo	Indicates whether or not browser information should be generated for the file.
	reserved	This field is reserved by the IDE.
	filetype	Returns the Mac OS file type of the file. This field is defined for Mac OS-hosted plug-ins only.
	filecreator	Returns the Mac OS creator signature of the file. This field is defined for Mac OS-hosted plug-ins only.
	hasunitdata	Indicates whether the file has associated unit data (Pascal).
	mergeintooutput	Returns true if the "Merge Into Output" checkbox is checked in the project inspector.
	unitdatadependencytag	Returns a dependency tag (checksum) of unit data (Pascal).
Remarks	<p><a href="#">CWGetFileInfo</a> uses this data structure to describe a file in the active project.</p> <p><code>filespec</code> returns the file specification for the specified file. The IDE will return the most recent known (cached) file location if the plug-in passes 0 for the <code>checkFileLocation</code> parameter when calling <a href="#">CWGetFileInfo</a>. Passing 1 tells the IDE to verify the file's current location on disk.</p> <p>The <code>hasobjectcode</code>, <code>hasresources</code>, and <code>isresourcefile</code> fields have very precise and not entirely intuitive meanings:</p> <ul style="list-style-type: none"> <li>• <code>hasobjectcode</code> means that the specified file is a source file and that compiling the source will produce object data.</li> <li>• <code>hasresources</code> means that the specified file is a source file, and compiling it will produce resource data.</li> <li>• <code>isresourcefile</code> means that the file itself is resource data, which should simply be copied into the target binary (this is typically only the case on Macintosh).</li> </ul>	

Note that neither `hasobjectcode` nor `hasresources` imply that such data actually exists for the file yet; plug-ins determine this by loading the object data using [CWLoadObjectData](#) and examining it (remember to call [CWFreeObjectData](#) afterward).

`dropinname` is the internal “plug-in name” of the plug-in used to process the file, not the plug-in’s display name.

Plug-ins use `fileID` when generating browse information. This file ID uniquely identifies a target file. Plug-ins should emit this value in browser records in fields that specify the source file for a symbol. See [“Browser Reference” on page 573](#) for more information.

`gendebbug` reflects the debug setting in the main project window. Compilers and linkers should only generate debug code for files which have this flag set.

**Mac OS** The `segment` field applies only to Mac 68K targets, and its value is determined by the assignment of files to segments in the segments tab of the project.

The `weakimport`, `initbefore`, and `mergeintooutput` flags apply only to library files included in Mac OS targets. The values of the flags match the values established in the project inspector window.

The `weakimport` flag indicates whether the corresponding checkbox is set in the project inspector window for this file, and when set indicates that the target application wishes to run even if it cannot bind to all imported library symbols (variables and routines) at launch time.

`mergeintooutput` indicates that the library should be copied into the final executable. This ensures that the library will be available at runtime (much like statically linking a dynamic library).

**Versions** In versions of the API prior to 11, the `objmoddate` field of this structure returned the modification date of the associated source file. It now correctly returns the date of last modification of the object code.

**See Also** [“CWGetFileInfo” on page 120](#)

## **CWProjectSegmentInfo**

**Description** Contains information about a segment in the active project.

**Definition**

```
#include <CWPlugins.h>
typedef struct CWProjectSegmentInfo {
    char           name[32];
    short          attributes;
} CWProjectSegmentInfo;
```

**Fields** The fields in `CWProjectFileInfo` are:

name The name of the segment

attributes The Resource Manager attributes for the segment.

**Remarks** [CWGetSegmentInfo](#) uses this data structure to describe a segment in the current project target, for Mac OS 68K targets only. Segments are blocks of compiled code, stored in ‘CODE’ resources in the Mac OS 68K runtime environment. The segment tab of a 68K target specifies which files will be placed in each ‘CODE’ resource.

A `CWProjectSegmentInfo` structure specifies the name and resource manager attributes for a segment (any combination of preload, locked, purgeable, protected, system heap).

**See Also** [“CWGetSegmentInfo” on page 132](#)

## **CWRelativePath**

**Description** Describes a directory specified relative to another.

**Definition**

```
#include <CWPlugins.h>
typedef struct CWRelativePath
{
    short          version;
    unsigned char  pathType;
    unsigned char  pathFormat;
    char           userDefinedTree[256];
    char           pathString[512];
} CWRelativePath;
```

**Fields** The fields in `CWRelativePath` are:

version	Specifies the version of the CWRelativePath record.
pathType	Specifies the reference directory that the path is relative to. This should be one of the <a href="#">CWRelativePathTypes</a> enumeration values: type_Absolute, type_Project, type_Compiler, type_System, type_UserDefined.
pathFormat	Specifies the OS path syntax used in pathString. This should be one of the <a href="#">CWRelativePathFormat</a> enumeration values: format_Generic, format_Mac, format_Win, format_Unix.
userDefinedTree	Specifies the path to the reference directory, when pathType is type_UserDefined.
pathString	Specifies a directory location, relative to the location specified by pathType and optionally userDefinedTree.

**Remarks** A CWRelativePath structure specifies a relative directory location. A relative path specifies a directory location using a sequence of subdirectories starting from another absolute reference location.

Usually, the reference location, specified by pathType, is a well-known location that typically exists on all IDE hosts. Examples of well-known locations include the directory containing the compiler and its support files, and the directory containing a project.

The partial path string stored in pathString may be formatted for any of the supported platforms.

Relative paths are especially useful when the reference path may have a different location on different host systems. Relative paths help to make directory specifications portable. They also eliminate the need to determine the locations of standard directories.

[CWPanlChooseRelativePath](#) returns CWRelativePath structures which may be resolved to full absolute paths using [CWResolveRelativePath](#).

See Also    [“CWResolveRelativePath” on page 142](#)

[“CWPanelChooseRelativePath” on page 420](#)

[“CWRelativePathFormat” on page 180](#)

[“CWRelativePathTypes” on page 181](#)

## **CWRelativePathFormat**

Description    Specifies the operating system format of a relative path.

Definition

```
#include <CWPlugins.h>
typedef enum CWRelativePathFormat
{
    format_Generic = 0,
    format_Mac,
    format_Win,
    format_UNIX
} CWRelativePathFormat;
```

Values    These constants have the following meanings:

format\_Generic    Indicates a simple file name, without any additional parts, such as drive or path specifiers.

format\_Mac    Indicates Mac OS path syntax. Directory names are separated by colon (‘:’) characters.

format\_Win    Indicates Windows path syntax. Directory names are separated by backward slash characters (‘\’).

format\_UNIX    Indicates Unix path syntax. Directory names are separated by forward slash characters (‘/’).

Remarks    This type specifies the format of a relative path in a [CWRelativePath](#) structure returned by [CWPanelChooseRelativePath](#).

See Also    [“CWResolveRelativePath” on page 142](#)

[“CWRelativePath” on page 178](#)

[“CWRelativePathTypes” on page 181](#)

[“CWPanelChooseRelativePath” on page 420](#)

## CWRelativePathTypes

Description	Describes an absolute location referred to by a relative path.	
Definition	<pre>#include &lt;CWPlugins.h&gt; typedef enum CWRelativePathTypes {     type_Absolute= 0,     type_Project,     type_Compiler,     type_System,     type_UserDefined } CWRelativePathTypes;</pre>	
Values	These constants have the following meanings:	
	type_Absolute	Indicates that the relative path is not actually relative, but instead specifies a full absolute path.
	type_Project	Indicates that the path is specified relative to the directory containing the project.
	type_Compiler	Indicates that the path is specified relative to the directory containing the compiler.
	type_System	Indicates that the path is specified relative to the directory containing the system.  On Windows, this refers to the Windows\System directory on the boot drive. On Mac OS, this is the System folder on the boot volume.
	type_UserDefined	Indicates that the relative path refers to a directory specified by the plug-in.
Remarks	This enumeration specifies the starting point for a relative path. Relative paths constructed by <a href="#">CWPanelChooseRelativePath</a> return a value of this type in the pathType field.	
See Also	<a href="#">“CWResolveRelativePath” on page 142</a>	

[“CWRelativePath” on page 178](#)

[“CWRelativePathFormat” on page 180](#)

[“CWPanelChooseRelativePath” on page 420](#)

## **CWResult**

Description      Specifies a result code returned by a call to the IDE.

Definition      

```
#include <CWPlugins.h>
typedef long CWResult;
```

Remarks      All routines provided by the plug-in API return a result of type CWResult. The most common errors are returned directly. For OS-specific errors, the CWResult is [cwErrOSError](#), and the OS-specific error can be obtained by calling [CWGetCallbackOSError](#).

See Also      [“cwErrOSError” on page 195](#)

[“CWGetCallbackOSError” on page 117](#)

## **DropInFlags**

Description      Describes a plug-in and its capabilities to the IDE.

Definition      

```
#include <CWPlugins.h>
typedef struct DropInFlags
{
    short          rsrcversion;
    CWDataType   dropintype;
    unsigned short earliestCompatibleAPIVersion;
    unsigned long  dropinflags;
    CWDataType   edit_language;
    unsigned short newestAPIVersion;
} DropInFlags, **DropInFlagsHandle;
```

Fields      The fields in DropInFlags are:

<b>rsrcversion</b>	Indicates the version number of the <code>DropInFlags</code> structure. Usually, new plug-ins will want to use the current version of the structure. Use <code>kCurrentDropInFlagsVersion</code> to indicate the latest version.
<b>dropintype</b>	Specifies the plug-in type. Should be one of <code>CWDROPINLINKERTYPE</code> , <code>CWDROPINCOMPILERTYPE</code> , <code>CWDROPINPREFSTYPE</code> , <code>CWDROPINVCSTYPE</code> , or <code>CWDROPINCOMTYPE</code> .
<b>earliestCompatibleAPIVersion</b>	Specifies the earliest API version with which the plug-in can operate properly.
<b>dropinflags</b>	Contains bit flags indicating plug-in capabilities. Flag meanings vary with each plug-in type.
<b>edit_language</b>	The source language of files accepted by the plug-in (where relevant).
<b>newestAPIVersion</b>	Specifies the most recent API version with which the plug-in can operate properly.
<b>Remarks</b>	<p>The <code>DropInFlags</code> structure is returned by compiler, linker, and version control plug-ins to the IDE via their <code>CWPlugin_GetDropInFlags</code> entry point. This structure provides the IDE with information about the plug-in and its capabilities.</p> <p>The <code>dropintype</code> field consists of bit flags having different meanings for each plug-in type. See <a href="#">Compiler Capability Flags</a>, <a href="#">Linker Capability Flags</a>, and <a href="#">VCS Plug-in Capability Flags</a> for more information. Preference panels use a similar but different structure to report their capabilities. See <a href="#">PanelFlags</a> for more information.</p> <p>The <code>newestAPIVersion</code> field should usually be set to <code>DROPINCOMPILERLINKERAPIVERSION</code> or <code>DROPINPANELAPIVERSION</code>, depending upon the type of the plug-</p>

in. Each update of the API headers sets the value of this define to the latest API version.

For most plug-ins, `earliestCompatibleAPIVersion` should be set to the latest version of the API that was in effect when the plug-in was first created. Usually, this should be done using a specific version define, rather than one of

`DROPINCOMPILERLINKERAPIVERSION` or  
`DROPINPANELAPIVERSION` (since their values change with API releases, but the earliest API version supported by a plug-in usually does not).

See Also

[“Compiler Capability Flags” on page 200](#)

[“Linker Capability Flags” on page 203](#)

[“CWPlugin\\_GetDropInFlags Entry Point” on page 150](#)

[“Specifying Plug-in Capabilities” on page 65](#)

## Constants for Plug-ins

This section describes the constant and defines used by the CodeWarrior Plug-in API.

These constants are:

- [CWDROPINCOMPILERTYPE](#)
- [CWDROPINLINKERTYPE](#)
- [CWDROPINPREFSTYPE](#)
- [CWDROPINPREFSTYPE\\_1](#)
- [CWDROPINVVCSTYPE](#)
- [cwFileTypePrecompiledHeader](#)
- [cwFileTypeText](#)
- [cwFileTypeUnknown](#)
- [kCurrentCompiledFile](#)
- [kDefaultLinkPosition](#)
- [kTargetGlobalPluginData](#)
- [messagetypeInfo](#)

- [messagetypeWarning](#)
- [messagetypeError](#)
- [reqAbout](#)
- [reqIdle](#)
- [reqInitialize](#)
- [reqPrefsChange](#)
- [reqTerminate](#)

## CWDROPINCOMPILERTYPE

Description	Defines the file type of compiler plug-ins.
Definition	<pre>#include "DropinCompilerLinker.h" enum {     /* ... */     CWDROPINCOMPILERTYPE = 'Comp',     /* ... */ };</pre>
Remarks	Use <b>CWDROPINCOMPILERTYPE</b> in the <code>dropintype</code> field of a <code>DropInFlags</code> structure to specify that a plug-in is a compiler.
Mac OS	This value should also be used for the file type of a compiler plug-in.

## CWDROPINLINKERTYPE

Description	Defines the file type of linker plug-ins.
Definition	<pre>#include "DropinCompilerLinker.h" enum {     /* ... */     CWDROPINLINKERTYPE = 'Link',     /* ... */ };</pre>
Remarks	Use <b>CWDROPINLINKERTYPE</b> in the <code>dropintype</code> field of a <code>DropInFlags</code> structure to specify that a plug-in is a linker.
Mac OS	This value should also be used for the file type of a linker plug-in.

## **CWDROPINPREFSTYPE**

Description	Defines the file type of settings panel plug-ins.
Definition	#include "DropinCompilerLinker.h" enum { /* ... */ CWDROPINPREFSTYPE      = 'PanL', /* ... */ };
Remarks	Use <b>CWDROPINPREFSTYPE</b> in the <b>dropintype</b> field of a <b>DropInFlags</b> structure to specify that a plug-in is a preference panel.
Mac OS	This value should also be used for the file type of a preference panel plug-in.

## **CWDROPINPREFSTYPE\_1**

Description	Defines the dropin type of settings panel plug-ins created for versions of the IDE before version 2.0.
Definition	#include "DropinCompilerLinker.h" enum { /* ... */ CWDROPINPREFSTYPE_1    = 'Panl', /* ... */ };
Remarks	<b>CWDROPINPREFSTYPE_1</b> was used in the <b>dropintype</b> field of a <b>DropInFlags</b> structure for 1.x versions of the IDE.
Mac OS	This value was also used for the file type of a version 1 preference panel plug-in.
<b>NOTE</b>	This type should not be used for new plug-ins. However, new versions of the IDE continue to support old preference panels.

## CWDROPINVCSTYPE

Description	Defines the file type of version control system (VCS) plug-ins.
Definition	<pre>#include "DropinCompilerLinker.h" enum {     /* ... */     CWDROPINVCSTYPE           = 'VCS'     /* ... */ };</pre>
Remarks	Use <code>CWDROPINVCSTYPE</code> in the <code>dropintype</code> field of a <code>DropInFlags</code> structure to specify that a plug-in is a version control system.
Mac OS	This value should also be used for the file type of a VCS plug-in.

## cwFileTypePrecompiledHeader

Description	Specifies that the IDE has found a cached precompiled header file.
Definition	<pre>#include "DropinCompilerLinker.h" enum {     /* ... */     cwFileTypePrecompiledHeader     /* ... */ };</pre>
Remarks	The IDE uses <code>cwFileTypePrecompiledHeader</code> in the <code>filedatatype</code> field of a <a href="#">CWFFileInfo</a> data structure to specify that it has found a precompiled header file that has been cached.
See Also	<a href="#">“CWFFileInfo” on page 159</a>

## cwFileTypeText

Description	Specifies that the IDE has found a text file.
Definition	<pre>#include "DropinCompilerLinker.h" enum {     /* ... */     cwFileTypeText,     /* ... */ };</pre>

Remarks	The IDE uses <code>cwFileTypeText</code> in the <code>filedatatype</code> field of a <a href="#">CWFileInfo</a> data structure to specify that it has found a precompiled header file that has been cached.
See Also	<a href="#">“CWFileInfo” on page 159</a>

## **cwFileTypeUnknown**

Description Specifies that the IDE has found a binary file.

Definition

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    cwFileTypeUnknown,
    /* ... */
};
```

Remarks The IDE uses `cwFileTypeUnknown` in the `filedatatype` field of a [CWFileInfo](#) data structure to specify that it has found a file that contains unknown binary data.

See Also [“CWFileInfo” on page 159](#)

## **cwSystemPath**

Description Specifies a system path as configured in the **Access Paths** panel.

Definition

```
#include <CWPlugins.h>
```

Remarks Specifies a system path in calls to `CWGetAccessPathInfo` and `CWGetAccessPathSubdirectory`. System paths are searched after user paths.

See Also [“CWOvlay1GroupInfo” on page 171](#)

## **cwUserPath**

Description Specifies a user path as configured in the **Access Paths** panel.

Definition

```
#include <CWPlugins.h>
```

FieldRemarks Specifies a user path in calls to `CWGetAccessPathInfo` and `CWGetAccessPathSubdirectory`. User paths are searched before system paths.

See Also [“CWOvlay1GroupInfo” on page 171](#)

## **kCurrentCompiledFile**

Description	Specifies that a file is dependent on the file that is currently being compiled by the IDE.
Definition	#include "DropinCompilerLinker.h" #define kCurrentCompiledFile -1L
Remarks	The IDE uses <code>kCurrentCompiledFile</code> in the <code>isdependentoffile</code> field of a <a href="#">CWFileInfo</a> data structure to specify that a file depends on the file that is currently being compiled.
See Also	<a href="#">“CWFileInfo” on page 159</a>

## **kDefaultLinkPosition**

Description	Specifies the default position in an ordering of project files.
Definition	#include "DropinCompilerLinker.h" #define kDefaultLinkPosition -1L
Remarks	Plug-ins use <code>kDefaultLinkPosition</code> in a <a href="#">CWNewProjectEntryInfo</a> structure for any of the <code>position</code> , <code>segment</code> , <code>overlayGroup</code> , and <code>overlay</code> fields, to specify that the IDE should insert the new file in the default position with the respective file ordering.
See Also	<a href="#">“CWNewProjectEntryInfo” on page 168</a>

## **kTargetGlobalPluginData**

Description	Specifies data that applies globally to a project's target rather than a single file.
Definition	#include "CWPlugins.h" #define kTargetGlobalPluginData -1L
Remarks	A plug-in passes <code>kTargetGlobalPluginData</code> in the <code>whichfile</code> parameters for <a href="#">CWStorePluginData</a> and <a href="#">CWGetPluginData</a> to specify that the IDE should store data globally for the active target rather than a single file in the target.
See Also	<a href="#">“CWStorePluginData” on page 145</a> <a href="#">“CWGetPluginData” on page 129</a>

## **messagetypeInfo**

Description     Specifies the nature of an item to appear in a message window.

Definition    

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    messagetypeInfo,
    /* ... */
};
```

Remarks      Use this constant in the `errorlevel` argument for [CWReportMessage](#) to specify that a message provides a piece of information that isn't an error or warning.

See Also     [“CWReportMessage” on page 140](#)

## **messagetypeWarning**

Description     Specifies the nature of an item to appear in a message window.

Definition    

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    messagetypeWarning,
    /* ... */
};
```

Remarks      Use this constant in the `errorlevel` argument for [CWReportMessage](#) to specify that a message is a warning.

See Also     [“CWReportMessage” on page 140](#)

## **messagetypeError**

Description     Specifies the nature of an item to appear in a message window.

Definition    

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    messagetypeError,
    /* ... */
};
```

Remarks	Use this constant in the <code>errorlevel</code> argument for <a href="#">CWReportMessage</a> to specify that a message is an error.
See Also	<a href="#">“CWReportMessage” on page 140</a>

## reqAbout

Description Asks the plug-in to display its about box.

Definition

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    reqAbout = -101
    /* ... */
};
```

Remarks This request is sent by the IDE to ask a plug-in to display its about box.

---

**NOTE** Currently, this request is only sent for version control plug-ins when the user selects **About** from the version control menu.

---

See Also [“CWGetPluginRequest” on page 130](#)

## reqIdle

Description Gives the plug-in a chance to perform a low priority background task.

Definition

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    reqIdle = -100
    /* ... */
};
```

Remarks The IDE sends this request occasionally (roughly once every two minutes), allowing a plug-in to perform some low priority background task. Plug-ins should avoid making assumptions about the exact frequency of `reqIdle` requests.

---

**NOTE** Currently, this request is only sent for version control plug-ins.

---

**WARNING!** Plug-ins may not make any other calls to the IDE when handling this request.

---

See Also [“CWGetPluginRequest” on page 130](#)

## **reqInitialize**

Description Asks the plug-in to set itself to a known initial state.

Definition 

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    reqInitialize = -2
    /* ... */
};
```

Remarks This is the first request issued from the IDE to a plug-in after the IDE has loaded it into memory. While handling a `reqInitialize` request, a plug-in may only call [CWGetPluginRequest](#) and [CWDonePluginRequest](#).

See Also [“CWGetPluginRequest” on page 130](#)

[“reqInitPanel” on page 509](#)

## **reqPrefsChange**

Description Informs the plug-in that its preferences have changed.

Definition 

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    reqInitialize = -102
    /* ... */
};
```

Remarks This message is sent when the user changes the preferences for a plug-in. Typically, a plug-in may wish to respond by reloading its preference data.

---

**NOTE** Currently, this request is only sent for version control plug-ins.

---

See Also [“CWGetPluginRequest” on page 130](#)

---

## reqTerminate

Description     Asks the plug-in to clean up before it is unloaded.

Definition    

```
#include "DropinCompilerLinker.h"
enum {
    /* ... */
    reqTerminate = -1
    /* ... */
};
```

Remarks      This is the last request issued from the IDE to the plug-in before the IDE unloads the plug-in from memory. While handling a `reqTerminate` request, a plug-in may only call [CWGetPluginRequest](#) and [CWDonePluginRequest](#).

See Also     [“CWGetPluginRequest” on page 130](#)  
[“reqTermPanel” on page 514](#)

## Result Codes for Plug-ins

This section lists error codes that the IDE recognizes. A plug-in should return a value in this section when handling a request from the IDE. The IDE returns a value from this section when a plug-in calls it.

These values are:

- [cwErrCantSetAttribute](#)
- [cwErrFileNotFoundException](#)
- [cwErrInvalidCallback](#)
- [cwErrInvalidMPCallback](#)
- [cwErrInvalidParameter](#)
- [cwErrOSError](#)
- [cwErrOutOfMemory](#)
- [cwErrRequestFailed](#)
- [cwErrSilent](#)
- [cwErrStringBufferOverflow](#)
- [cwErrUnknownFile](#)

- [cwErrUserCanceled](#)
- [cwNoErr](#)

**NOTE** The literal values listed in this section are for reference only (useful during debugging, for example), and do not appear in the headers. Always use constants rather than the literal values in plug-in code.

## **cwErrCantSetAttribute**

Description The plug-in requested inappropriate flags in [CWAddProjectEntry](#).

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrCantSetAttribute = 11,
    /* ... */
};
```

## **cwErrFileNotFoundException**

Description A file was not found on disk.

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrFileNotFoundException = 8,
    /* ... */
};
```

## **cwErrInvalidCallback**

Description The IDE isn't able to provide a routine for a plug-in.

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrInvalidCallback = 4,
```

```
    /* ... */  
};
```

Remarks	The IDE issues this error code when a plug-in tries to call an IDE routine that isn't appropriate for the request the plug-in is acting on or the plug-in's type.
---------	---

## **cwErrInvalidMPCallback**

Description	The plug-in can't act on a request while executing as a multiprocessing thread.
-------------	---

```
#include <CWPluginErrors.h>  
enum  
{  
    /* ... */  
    cwErrInvalidMPCallback = 5,  
    /* ... */  
};
```

## **cwErrInvalidParameter**

Description	An IDE routine can't complete because a parameter passed to it is bad.
-------------	--

```
#include <CWPluginErrors.h>  
enum  
{  
    /* ... */  
    cwErrInvalidParameter = 3,  
    /* ... */  
};
```

## **cwErrOSError**

Description	The host operating system issued an error.
-------------	--

```
#include <CWPluginErrors.h>  
enum  
{  
    /* ... */  
    cwErrOSError = 6,  
    /* ... */  
};
```

};

## **cwErrOutOfMemory**

Description    There was not enough memory available to complete a memory allocation.

Definition    

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrOutOfMemory = 7,
    /* ... */
};
```

## **cwErrRequestFailed**

Description    The plug-in couldn't service a request successfully.

Definition    

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrRequestFailed = 2,
    /* ... */
};
```

## **cwErrSilent**

Description    An error code that a plug-in may pass back to the IDE when a request failed, but the plug-in doesn't want the IDE to report the error. This is useful when the IDE's error reporting is inadequate or misleading, and the plug-in reports the error instead.

Definition    

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrSilent = 10,
    /* ... */
};
```

## **cwErrStringBufferOverflow**

Description An output string buffer was too small.

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrStringBufferOverflow = 12,
    /* ... */
};
```

## **cwErrUnknownFile**

Description An invalid file number was used.

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrUnknownFile = 9,
    /* ... */
};
```

## **cwErrUserCanceled**

Description The user has issued a command to stop the current operation.

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
    cwErrUserCanceled = 1,
    /* ... */
};
```

## **cwNoErr**

Description An operation was completed successfully.

Definition

```
#include <CWPluginErrors.h>
enum
{
    /* ... */
}
```

```
    cwNoErr = 0,  
    /* ... */  
};
```

# Creating Compiler and Linker Plug-ins

---

This chapter shows you how to create compiler, linker, pre-linker, and post-linker plug-ins for the CodeWarrior IDE. It discusses issues and API services specific to compilers and linkers.

## Overview

This chapter describes the parts of a plug-in, how a plug-in communicates with the IDE, and how it should perform its tasks.

This chapter covers the following topics:

- [Compiler and Linker DropIn Flags](#)
- [Compiler and Linker-Specific Entry Points](#)
- [Handling Compiler and Linker Requests](#)
- [Managing Projects and Targets](#)

## Compiler and Linker DropIn Flags

The IDE requires that all plug-ins provide information about their capabilities. This is done through informational entry points.

As described in [Specifying Plug-in Capabilities](#), each plug-in must inform the IDE of its capabilities by returning a `DropInFlags` structure from its `CWPlugin_GetDropInFlags` entry point. The meanings of the bits in the `dropinflags` member vary with each plug-in type.

This section documents the meanings of the flags used by compilers and linkers. The topics covered in this section are:

- [Compiler Capability Flags](#)
- [Linker Capability Flags](#)

## Compiler Capability Flags

The value of `dropinflags` returned by compiler plug-ins consists of the bitwise-or (or arithmetic sum) of the following constants:

Flag name	Meaning
<code>kGeneratescode</code>	Indicates that the compiler generates compiled object code (some compilers may generate source code instead).
<code>kGeneratesrsrscs</code>	Indicates that the compiler output consists of binary resources.
<code>kCanpreprocess</code>	Indicates that the compiler supports preprocessing and wants to receive compile requests when the user selects <b>Project &gt; Preprocess</b> .
<code>kCanprecompile</code>	Indicates that the compiler supports precompilation and wants to receive compile requests when the user selects <b>Project &gt; Precompile</b> .
<code>kIspascal</code>	Used by the Metrowerks Pascal compiler; should not be used by other plug-ins.
<code>kCanimport</code>	Enables the <b>Weak Link</b> checkbox for file types associated with the plug-in.
<code>kCandisassemble</code>	Indicates that the plug-in wishes to receive the <code>reqCompDisassemble</code> request.
<code>kPersistent</code>	Tells the IDE to reload the plug-in as infrequently as possible. Setting this does <i>not</i> guarantee that the plug-in will remain resident for the duration of one IDE session.
<code>kCompAllowDupFileNames</code>	

<b>Flag name</b>	<b>Meaning</b>
	Indicates that the compiler properly handles different source files having the same name (but different paths) in one project.
kCompMultiTargAware	Currently unused. Do not set.
kIsMPAware	Currently unused. Do not set.
kCompUsesTargetStorage	Indicates that the plug-in expects to use <a href="#">CWGetTargetStorage</a> and <a href="#">CWSetsTargetStorage</a> to store data with the current target.
kCompEmitsOwnBrSymbols	Indicates that the plug-in emits its own custom types of browse symbols.
kCompAlwaysReload	Tells the IDE to reload the compiler before every request. Typically used to restore global data to a known state.
kCompRequiresProjectBuildStartedMsg	Indicates that the compiler wants the reqProjectBuildStarted and reqProjectBuildEnded messages from the IDE.
kCompRequiresTargetBuildStartedMsg	Indicates that the compiler wants the reqTargetBuildStarted and reqTargetBuildEnded messages from the IDE.
kCompRequiresSubProjectBuildStartedMsg	

## **Creating Compiler and Linker Plug-ins**

### *Compiler Capability Flags*

---

<b>Flag name</b>	<b>Meaning</b>
	Indicates that the compiler wants the <code>reqSubProjectBuildStarted</code> and <code>reqSubProjectBuildEnded</code> messages from the IDE.
<code>kCompRequiresFileListBuildStartedMsg</code>	Indicates that the compiler wants the <code>reqFileListBuildStarted</code> and <code>reqFileListBuildEnded</code> messages from the IDE.
<code>kCompReentrant</code>	Indicates that the IDE can call this compiler simultaneously from multiple threads.
<code>kCompSavesDbgPreprocess</code>	Obsolete. Do not use this flag.
<code>kCompRequiresTargetCompileStartedMsg</code>	Indicates that the compiler can receive target compile started and ended messages. Such a compiler can benefit from pre-work or post-processing after all individual compilations have been completed.

These flags are defined in the “`CompilerMapping.h`” header file. On Mac OS, these flags can be provided in a [“Flag Resource.”](#)

---

**NOTE** All unused flags should be set to zero, and are reserved by Metrowerks for future use. Also, be careful to avoid using similarly named flags for other tool types to specify compiler capabilities. For example, `kCompMultiTargAware` and `linkMultiTargAware` have similar names but different values and thus are not interchangeable.

---

## Linker Capability Flags

The value of `dropinflags` returned by compiler plug-ins consists of the bitwise-or (or arithmetic sum) of the following constants:

Flag name	Meaning
<code>cantDisassemble</code>	Indicates that linker does <i>not</i> support disassembly of its target executable code.
<code>isPostLinker</code>	Tells the IDE to call this tool after the link phase has completed. That is, this tool wants a <code>reqLink</code> message after the target executable has been linked.
<code>linkAllowDupFileNames</code>	Indicates that the linker properly handles different source files having the same name (but different paths) in one project.
<code>linkMultiTargAware</code>	When <i>clear</i> , the IDE will reload the linker prior to sending it a <code>reqLink</code> request for a target that differs from the last target linked. That is, the IDE will reload the linker every time the current target changes.
<code>isPreLinker</code>	Tells the IDE to call this tool after the compile phase has completed, and before linking begins. That is, this tool wants a <code>reqLink</code> message before target linking begins.
<code>linkerUsesTargetStorage</code>	Indicates that the linker uses <a href="#"><u>CWGetTargetStorage</u></a> and <a href="#"><u>CWSetTargetStorage</u></a> to store global data.

## **Creating Compiler and Linker Plug-ins**

### *Linker Capability Flags*

---

<b>Flag name</b>	<b>Meaning</b>
<code>linkerUnmangles</code>	Indicates that the linker exports an unmangling entry point that supports unmangling symbol names generated by the linker.
<code>magicCapLinker</code>	Obsolete; do not use.
<code>linkAlwaysReload</code>	Tells the IDE to reload the linker before every request. Typically used to restore global data to a known state.
<code>linkRequiresProjectBuildStartedMsg</code>	Indicates that the linker wants the <code>reqProjectBuildStarted</code> and <code>reqProjectBuildEnded</code> messages from the IDE.
<code>linkRequiresTargetBuildStartedMsg</code>	Indicates that the linker wants the <code>reqTargetBuildStarted</code> and <code>reqTargetBuildEnded</code> messages from the IDE.
<code>linkRequiresSubProjectBuildStartedMsg</code>	Indicates that the linker wants the <code>reqSubProjectBuildStarted</code> and <code>reqSubProjectBuildEnded</code> messages from the IDE.
<code>linkRequiresFileListBuildStartedMsg</code>	Indicates that the linker wants the <code>reqFileListBuildStarted</code> and <code>reqFileListBuildEnded</code> messages from the IDE.
<code>linkRequiresTargetLinkStartedMsg</code>	Indicates that the linker wants the <code>reqTargetLinkStarted</code> and <code>reqTargetLinkEnded</code> messages from the IDE.

<b>Flag name</b>	<b>Meaning</b>
linkerWantsPreRunRequest	Indicates that the linker or post-linker) wants to receive reqPreRun requests, sent by the IDE prior to launching the target executable. Does not apply to pre-linkers.
linkerGetTargetInfoThreadSafe	Indicates that the linker can correctly service the reqTargetInfo request while other requests, such as reqLink, are pending.
linkerUsesCaseInsensitiveSymbols	Indicates that the symbol browser ignores case when looking for symbols within this target.
linkerDisasmRequiresPreprocess	Obsolete. Do not use this flag.
linkerUsesFrameworks	Indicates that the target uses frameworks and enables framework-style file searching. This flag is significant only when generating Mach-O targets on the Mac OS.
linkerSuggestsNonRecursiveAccessPaths	Indicates that the linker should use non-recursive paths when adding a new project or file. This can save time and resources when searching and building.
linkerInitializeOnMainThread	Indicates that the linker needs to be initialized on the main thread.

---

These flags are defined in the “CompilerMapping.h” header file. On Mac OS, these flags can be provided in a [“Flag Resource.”](#)

---

**NOTE** All unused flags should be set to zero, and are reserved by Metrowerks for future use. Also, be careful to avoid using similarly named flags for other tool types to specify linker capabilities. For example, kCompMultiTargAware and linkMultiTargAware have similar names but different values and thus are not interchangeable.

---

## Compiler and Linker-Specific Entry Points

In addition to the standard informational entry points provided by all plug-ins, compilers and linkers also define two additional entry points, for indicating associated files, and supported target platforms. Compilers and linkers may also optionally provide entry points for generating browser symbol information, and for unmangling symbol names.

This section covers the following informational entry points:

- [Specifying File Mappings](#)
- [Specifying Target Platforms](#)
- [Compiler Browser Symbol Entry Point](#)
- [Linker Symbol Unmangling Entry Point](#)

### Specifying File Mappings

To specify the types of files that a plug-in can process, use:

```
CWPLUGIN_ENTRY (CWPlugin_GetDefaultMappingList)
    (const CWExtMapList**);
```

This entry point returns a list of items. The list consists of a version number, a count of mappings, and a pointer to an array of CWExtensionMapping structs. Each CWExtensionMapping struct specifies a file type, its corresponding extension, and flags indicating how the IDE should treat files of the specified type. See [Listing 6.1](#) for an example.

### **Listing 6.1 CWPlugin\_GetDefaultMappingList example**

```
CWPLUGIN_ENTRY(CWPlugin_GetDefaultMappingList)
    (const CWExtMapList** defaultMappingList)
{
    static CWExtensionMapping sExtension = {'TEXT', ".c", 0};
    static CWExtMapList sExtensionMapList =
{kCurrentCWExtMapListVersion, 1, &sExtension};

    *defaultMappingList = &sExtensionMapList;

    return cwNoErr;
}
```

---

The IDE adds the entries in a plug-in's mapping list to the current project whenever a project is created from scratch, or when the user manually reverts to factory settings. File mappings are *not* updated either when a file is opened, or when creating a new project based upon stationery.

The flags field of a CWExtensionMapping structure consists of the logical or, or sum, of the following flags:

- **kPrecompile**: this file type can be precompiled. If the associated compiler support precompilation, it will receive precompile requests for this file type.
- **kLaunchable**: this file type is launchable, meaning that the IDE will attempt to open the file with an appropriate application when the user double-clicks the file in the project window.
- **kRsrcfile**: this file type is a binary resource file containing resources to be copied into a final executable. Typically only used on Mac OS.
- **kIgnored**: this file type is ignored by the IDE during the build process. That is, the IDE will not process files of this type in any way.

A plug-in's mapping list should only list files that the user may add to a project's target. Files not usually added to a project, such as interfaces files, should not be listed in a plug-in's mapping list.

Mac OS      Alternately, you can define an '[EMap Resource](#)' to specify the types of files that the plug-in can process.

**NOTE** Use of `CWPlugin_GetDefaultMappingList` overrides the use of an '`Emap`' resource.

---

## Specifying Target Platforms

To specify the processor and operating system that a plug-in generates software for (its target), use:

```
CWPLUGIN_ENTRY (CWPlugin_GetTargetList)
    (const CWTtargetList** targetList)
```

This entry point returns two lists, one containing 4-byte codes indicating the processor that a plug-in generates software for, and another containing 4-byte codes that represent the operating system for which the plug-in generates software. See [Listing 6.2](#) for an example.

### **Listing 6.2** CWPlugin\_GetTargetList example

```
CWPLUGIN_ENTRY (CWPlugin_GetTargetList)
    (const CWTtargetList** targetList)
{
    static CWDatatype sCPU = 'SAMP';
    static CWDatatype sOS = 'SAMP';
    static CWTtargetList sTargetList = {kCurrentCWTtargetListVersion,
        1, &sCPU, 1, &sOS};

    *targetList = &sTargetList;

    return cwNoErr;
}
```

---

The header files that ship with the plug-in SDK define 4-byte codes that should be used for standard OSes and CPUs. If a plug-in's output can be used on any processor or platform, specify '\*\*\*\*\*' as the plug-in's target processor or platform.

**NOTE** Metrowerks reserves for future use all 4-byte CPU and OS codes that consist entirely of lowercase characters.

---

Mac OS You can use a “["Targ' Resource"](#) to specify the same information as the CWPLUGIN\_ENTRY routine.

---

**NOTE** Use of CWPlugin\_GetTargetList overrides use of a 'Targ' resource.

---

## Compiler Browser Symbol Entry Point

Most compilers extract symbolic information from source files during compilation. Compilers that plug in to the IDE may optionally return this information to the IDE. This information is known as “browser symbol information.”

The plug-in API defines the [EBrowserItem](#) type and a corresponding anonymous enumeration which specify standard types of browser symbols. Browser windows displayed in the IDE list all available symbol types and allow the user to select the type of symbol to view.

Compilers can extend the types of symbols displayed by the IDE. For example, an Objective-C compiler might store browser information for “protocols.” Since there is no predefined symbol type for protocols, this would be accomplished by assigning each protocol symbol a custom browser symbol code. Custom browser symbol code values reserved for use by plug-in compilers range from browseCompSymbolStart to browseEnd. For more information about the format of browse information, see [“Browser Reference” on page 573](#).

Compilers that emit custom browse symbol types should implement an additional entry point for returning browser symbol type names to the IDE. This entry point is declared as follows:

```
CWPLUGIN_ENTRY (Helper_GetCompilerBrSymbols)  
(CWCompilerBrSymbolInfo*);
```

This entry point is called when the IDE displays a browser window for a project that uses the plug-in. This entry point is only called if the compiler sets the [kCompEmitsOwnBrSymbols](#) flag in its [DropInFlags](#) structure. See [Linker Capability Flags](#) and [Specifying Plug-in Capabilities](#) for more information about linker flags and plug-in flags in general.

Helper\_GetCompilerBrSymbols returns a list of name pairs for each browse symbol type it defines (see [CWCompilerBrSymbol](#)).

The names specify an internal name for each symbol type and a user-interface name. The IDE groups symbolic information from all compilers used in the current target according to the internal name. The IDE displays symbol type names using the user interface name.

- Mac OS 68K resource-based plug-ins provide a '[Dhlp' Resource](#)' that implements this entry point as its main function.

## Linker Symbol Unmangling Entry Point

Linkers may provide an optional entry point for unmangling names. Unmangling is the process of converting names used in object code to support type-safe linking back into human-friendly names resembling source code.

The symbol unmangling entry point should be declared as follows:

```
CWPLUGIN_ENTRY (Helper_Unmangle)  
(CWUnmangleInfo*);
```

This entry point is called for each symbol the IDE needs to unmangle. Unmangling occurs when the browser window is opened; browser symbols are mangled, as stored in browser data, and must be unmangled for human display. This entry point is only called if the plug-in sets the [linkerUnmangles](#) flag in its [DropInFlags](#) structure.

The IDE passes the plug-in a [CWUnmangleInfo](#) structure, which identifies the name and type of the symbol to be unmangled, and provides a buffer in which to place the unmangled name.

Some linkers may find it helpful to store global state related to symbol unmangling. The IDE supports this directly using the targetStorage field of the [CWUnmangleInfo](#) structure. When the plug-in receives a [reqTargetLoaded](#) request, it should store any needed state using [CWSetTargetStorage](#). The IDE passes this state to the plug-in every time it calls the Helper\_Unmangle entry point, to assist in unmangling.

# Handling Compiler and Linker Requests

In addition to the basic requests which the IDE sends to all plug-ins (see “[Responding to IDE Requests](#)” on page 75), the IDE also sends plug-ins specialized requests specific to their type. This section describes the following types of requests sent to compiler and linker plug-ins:

- [Compiling](#)
- [Linking](#)
- [Providing Target Information](#)
- [Disassembling](#)
- [Generating Browser Data](#)
- [Generating Debugging Data](#)
- [Additional Compiler and Linker Requests](#)

**NOTE** In addition to the requests discussed here, compilers may be called by the IDE to return information about custom browser symbols, and linkers may be called by the IDE to unmangle symbols. These optional requests are made via separate entry points, rather than the main entry point. For more information, see “[Compiler Browser Symbol Entry Point](#)” on page 209 and “[Linker Symbol Unmangling Entry Point](#)” on page 210.

---

## Compiling

The IDE sends the `reqCompile` request to tell a compiler plug-in to compile a file in the active project. In response to this, most compilers should load the text of the current source file, compile the source text, generate object code (possibly including debugging information), and store it in the project.

The IDE supports several types of compilation:

- normal compilation (**Project > Compile**, **Project > Check Syntax**, **Project > Bring Up To Date**, or **Project > Make**)
- preprocessing (**Project > Preprocess**)
- precompilation (**Project > Precompile**)

In response to all of these user commands, the IDE issues a compile request to the plug-in ([reqCompile](#)). The plug-in determines the specific type of compile by asking the IDE for further information about the current request.

## Preprocessing

The plug-in calls [CWIsPreprocessing](#) to determine if the file to process should be preprocessed rather than compiled.

[CWIsPreprocessing](#) returns true when the user selects **Project > Preprocess**. The IDE will only send preprocessing requests to plug-ins which ask for them by setting the `kCanpreprocess` bit in their plug-in flags (see [Compiler Capability Flags](#)).

Among common languages, preprocessing is best defined for C and C++, but in general involves performing textual substitution and expansion that would normally occur before compilation. Usually, plug-ins should display the textual result of preprocessing to the user in a window, using [CWCreateNewTextDocument](#).

## Precompiling

A plug-in calls [CWIsPrecompiling](#) and [CWIsAutoPrecompiling](#) to determine if the file should be precompiled instead of compiled. [CWIsPrecompiling](#) returns true when the user selects **Project > Precompile** or when the current file is being automatically precompiled in response to the user selecting **Project > Make**. [CWIsAutoPrecompiling](#) returns true only when the user selects **Project > Make** and a source file that produces a precompiled file must be updated.

The IDE will only send precompilation requests (either manual or automatic) to plug-ins that ask for them, by setting the `kCanprecompile` bit in their plug-in flags (see [Compiler Capability Flags](#)). In both cases, the compiler plug-in should normally write a file containing precompiled symbolic information to disk. The plug-in should call [CWGetPrecompiledHeaderSpec](#) to determine where to store the precompiled data. Precompiled files are for use by plug-ins only, not the IDE, and consequently the format of a precompiled data file is completely up to a plug-in.

## Normal Compilation

If none of [CWIsPreprocessing](#), [CWIsPrecompiling](#), and [CWIsAutoPrecompiling](#) returns true, then the compile request should be treated normally. This will be the case if the user selects **Project > Compile**, **Bring Up To Date**, **Make**, or **Check Syntax**. The plug-in should compile the current source file and add its object code to the project using [CWStoreObjectData](#).

Note that there is currently no way to distinguish the **Check Syntax** operation. The API headers declare a `reqCheckSyntax` request, but this is not currently issued by the IDE. Thus, plug-ins currently must generate object code and store it, even though the IDE discards it when the plug-in finishes handling the compile request.

---

**NOTE** To enable concurrent compilation, the compiler must have the `kCompReentrant` flag and be code thread-safe.

---

## Obtaining Source Text

To compile a source file, a plug-in must load its text into memory. There are at least two ways a plug-in may obtain the text of a file being compiled:

- [CWGetMainFileText](#) loads the contents of the file currently being compiled into memory, and returns a pointer to the text.
- [CWGetMainFileSpec](#) retrieves the file specification of the file to process. With this file specification, the plug-in can use [CWGetFileText](#) to load the file into memory.

The plug-in can also retrieve other information about the file to process. [CWGetMainFileName](#) retrieves the index of the file to process within the link order of the active project target. Use this file number when storing object data using [CWStoreObjectData](#). [CWGetMainFileID](#) retrieves the file ID of the file to process. Use this file ID when storing browser data.

## Linking

The IDE supports three different types of linker plug-ins: linkers, pre-linkers, and post-linkers. Although linkers, pre-linkers, and post-linkers receive the same requests and have access to the same

API routines, the IDE calls them at different times during the build process.

- **Pre-linkers** typically modify existing object data or add new files based on object data generated from files in the current target before the linker does. Pre-linkers are called *before* all target object code has been bound into a single executable or library by a linker.
- **Linkers** typically generate an output file from object data.
- **Post-linkers** typically modify the linker's output, or do something with it, such as download it to a debugging host, reformat the code for burning into ROM, or merge multiple executables into a single package. Post-linkers are called *after* all target object code has been bound into a single executable or library by a linker.

The IDE sends the [reqLink](#) request to linker, pre-linker, and post linker plug-ins. In the case of pre-linkers and post-linkers, the action taken in response to this request may not resemble traditional linking.

### Linkers

To link all the object code for a target into a final binary executable or library, a linker calls [CWGetProjectFileCount](#) to get the number of files in the active project target. With this file count, the linker iterates through the project using [CWGetFileInfo](#) to determine which files in a project's target contain object or resource data.

To retrieve the object or resource data for a file in the current target, use [CWLoadObjectData](#). After processing a file's data, dispose the data using [CWFreeObjectData](#).

---

<b>NOTE</b>	The format of the “object data” stored and retrieved by <a href="#">CWStoreObjectData</a> and <a href="#">CWLoadObjectData</a> (usually object code or resource data) is determined by agreement between compiler and linker. It is left to the compiler and linker to agree on conventions that allow the linker to determine, for example, whether the stored data contains object code or resources (or both), to be bound into the target binary.
-------------	---

---

[Listing 6.3](#) shows an example framework for handling a [reqLink](#) request.

### **Listing 6.3 Handling a reqLink request for linkers**

```
static CWResult MyLink(CWPluginContext context)
{
    PrefStruct      prefsData;
    long            index;
    CWResult        err;
    long            filecount;

    err = MyGetPrefs(context, &prefsData);
    if (err != cwNoErr)
        return (err);
    err = CWGetProjectFileCount(context, &filecount);
    if (err != cwNoErr)
        return (err);
    for (index = 0; (err == noErr) && (index < filecount); index++)
    {
        CWProjectFileInfo   fileInfo;
        CWMemHandle         objectData;

        /* first, get info about the file */
        err = CWGetFileInfo(context, index, false, &fileInfo);
        if (err != noErr)
            continue;

        /* determine if we need to process this file */
        if ( !fileInfo.hasobjectcode &&
            !fileInfo.hasresources &&
            !fileInfo.isresourcefile)
            continue;

        if (fileInfo.isresourcefile)
        {
            /* handle binary resource files here */
        }
        else
        {
            /* handle object and resource data data */
        }
    }
}
```

```
/* load the object or resource data */
err = CWLoadObjectData(context, index, &objectData);
if (err != noErr)
    continue;
if (fileInfo.hasobjectcode)
{
    /* link the object code */
}
if (fileInfo.hasresources)
{
    /* copy resources */
}
/* release the object code when done */
err = CWFreeObjectData(context, index, objectData);
}
}
return (err);
}
```

---

### Pre-linkers

Pre-linkers generally take some action prior to linking, such filtering object code, or adding instrumentation code to a target. If a pre-linker adds new files to the project, it must also store object data for those files, since the compiler will not recompile any files added at the pre-link stage.

### Post-linkers

To get access to the output file and other information generated by the linker, a post-linker calls

- [CWGetTargetInfo](#) to get information about the linker's output file
- [CWGetOutputFileDirectory](#) to get information about the directory where the linker stored its output

## Providing Target Information

The IDE sends the [reqTargetInfo](#) request to query the active target's linker and post-linker plug-ins for information about their output. Linkers and post-linkers respond by filling out the fields of a

[CWTarGetInfo](#) data structure and passing it to the IDE by calling [CWSetTargetInfo](#). [Listing 6.4](#) shows an example of responding to a [reqTargetInfo](#) request.

A post-linker may modify the description of the linker's output by calling [CWGetTargetInfo](#) to retrieve details about the linker's output, modifying values in the [CWTarGetInfo](#) structure, and then submitting the changes to the IDE by calling [CWSetTargetInfo](#). In all subsequent calls to [CWGetTargetInfo](#), changes in target settings made by post-linkers supersede the information originally returned by linkers.

[Listing 6.4](#) provides a simple example of how to handle the [reqTargetInfo](#) request.

#### **Listing 6.4 Providing target information**

```
static CWResult MySetTargetInfo(CWPluginContext context)
{
    CWTarGetInfo targ;
    SamplePref prefsData;
    char* cstr;
    CWResult err;

    memset(&targ, 0, sizeof(targ));
    err = CWGetOutputFileDirectory(context, &targ.outfile);
    targ.outputType = linkOutputFile;
    targ.symfile = targ.outfile;
    targ.runfile = targ.outfile;
    targ.linkType = exelinkageFlat;
    targ.debuggerCreator = kMyDebuggerCreator;

    /* load the relevant prefs */
    err = MyGetPrefs(context, kSamplePanelName, &prefsData);
    if (err != cwNoErr)
        return (err);

    /* set other fields in targ based on prefs here */

    /* tell the IDE about the target */
    err = CWSetTargetInfo(context, &targ);
```

```
    return (err);  
}
```

---

## Specifying Runtime Options Using CWSetTargetInfo

In addition to describing linker output, the [CWTargetInfo](#) structure specifies runtime options which affect the way a linker's target is run and debugged.

When the user selects **Project > Run** or **Project > Debug**, the IDE launches either the linker's target executable, as specified by `outfile`, or a run or debug helper application, as specified by `runHelperName` or `debugHelperName`, on Windows, or by `runHelperCreator` or `debuggerCreator`, on Mac OS. When a helper application is specified, the IDE launches the helper application, and passes it the file specified by `runfile` plus any command line arguments specified by the plug-in in `args`. When no helper application is specified, the IDE launches the target executable, and passes it the `runfile`.

A plug-in is responsible for constructing the `args` string. Usually, a plug-in will call [CWGetCommandLineArgs](#) to obtain any parameters specified by the user in the **Runtime Settings** panel, and will pass them to the run or debug helper application in the `args` field along with any additional parameters needed to run the target executable.

Since the command line arguments obtained through [CWGetCommandLineArgs](#) are intended for use by the target executable, not the helper application, the linker typically encapsulates this information in a single command line parameter (by enclosing the parameters in quotes or some other suitable character), or by writing the information to be passed to the target executable to a temporary file. In the latter case, the linker must also inform the helper application of the temporary file's location, usually by passing an additional command line argument specifying its location.

Linkers may also wish to pass the user-specified working directory and environment variables to the helper application, since it is the process which constructs the final target executable's runtime environment. These can be passed to the run helper in a similar

fashion: either on the run helper's command line or in a temporary file. For more information about obtaining runtime settings, see [Getting Runtime Settings](#).

### Threading Issues and `reqTargetInfo`

Because the IDE issues the `reqTargetInfo` request fairly frequently, and because this request can be made while a plug-in is handling other requests (possibly including another `reqTargetInfo` request) from the IDE, it is important to handle the `reqTargetInfo` request in a reentrant and thread-safe fashion, if possible.

To handle the request safely, ensure the following:

1. All temporary computation done by the entry point should be performed using local, stack-allocated ("automatic" storage class) memory.
2. All access to global data should be bracketed with a synchronization primitive, such as a mutual exclusion ("mutex") semaphore, used by all parts of the plug-in code that access the global data.

In addition, the linker or post-linker plug-in should be sure to set the `linkerGetTargetInfoThreadSafe` bit in its [Linker Capability Flags](#).

## Disassembling

The IDE sends disassembly requests to both compilers and linkers. The literal value of the request is different for compilers and linkers, but the meaning of the request is the same. The IDE sends the `reqCompDisassemble` request to compiler plug-ins and the `reqDisassemble` request to linker plug-ins to tell plug-ins to convert object or resource data into a human-readable text format.

To determine which file to disassemble, a plug-in calls `CWGetMainFileNumber` which returns the file's number within the active project target. The plug-in then calls `CWGetFileInfo` to determine if the file generates object or resource data to disassemble. If the file has object or resource data, the plug-in can load the data using `CWLoadObjectData`. After processing the data, the plug-in should dispose it using `CWFreeObjectData`.

**NOTE** The hasobjectcode and hasresources fields of the [CWProjectFileInfo](#) structure returned by [CWGetFileInfo](#) do *not* indicate presence or absence of object and resource data. Instead, they indicate that such data could be present as a by-product of compiling the source file. To determine if such data is in fact present, a plug-in must load and examine the “object” data (which may not be present, or which may contain any combination of object code and resource data).

---

plug-ins normally store the disassembly text in a [CWMemHandle](#), fill out a [CWNewTextDocumentInfo](#) structure, and call [CWCreateNewTextDocument](#) to display the disassembled text in an editor window. [Listing 6.5](#) shows an example of how a linker plug-in handles a reqDisassemble request.

### **Listing 6.5 Handling a reqDisassemble or reqCompDisassemble request**

---

```
static CWResult MyDisassemble(CWPluginContext context)
{
    CWProjectFileInfo fileInfo;
    CWMemHandle objectData;
    size_t bufflen;
    CWResult err;
    long   fileNum;
    CWMemHandle output;

    /* get info about the file we have to disassemble */
    err = CWGetMainFileNumber(context, &fileNum);
    if (err != noErr)
        return (err);
    err = CWGetFileInfo(context, fileNum, false, &fileInfo);
    if (err == noErr)
    {
        if (fileInfo.hasobjectcode || fileInfo.hasresources)
        {
            /* load the object code or resource fork image */
            err = CWLoadObjectData(context, fileNum, &objectData);
            if (err == noErr)
            {
                /* process the object code or resources here */

```

```
/* resources can be disassembled into high-level */
/* resource descriptions, or they can be ignored */

/* release the object code when we're done */
err = CWFreeObjectData(context, fileNum, objectData);
}
else
    objectSize = 0;
}
else
{
    /* The file contains no object code or resources. */

}
err = CWAllocMemHandle(context, bufflen, true, &output);
if (err == cwNoErr)
{
    void* p;
    err = CWLockMemHandle(context, output, false, &p);
    if (err == noErr)
    {
        CWNewTextDocumentInfo docinfo;
        memcpy(p, buff, bufflen);
        CWUnlockMemHandle(context, output);
        memset(&docinfo, 0, sizeof(docinfo));
        docinfo.text = output;
        err = CWCreateNewTextDocument(context, &docinfo);
    }
}
return (err);
}
```

---

Note that disassembly of resource data is possible. A plug-in could “de-rez” binary resources back into compilable text form. Most often, however, disassemblers ignore resource data. Again, it is up to the compiler and linker to agree upon the format of object and resource data, so that it can be analyzed or ignored during disassembly.

## Generating Browser Data

Compilers which produce browser information in response to a compile request should call [CWGetBrowseOptions](#) to determine which types of browser information to generate. The IDE assumes that all plug-ins which generate browser information will always emit information for global variables and functions. The flags returned by [CWGetBrowseOptions](#) control the generation of information for additional language constructs such as classes, enumerations, macros, typedefs, constants, and templates.

Compilers can also emit their own custom types of browser symbols, by using symbol type codes starting with the value `browseCompSymbolStart`. Compilers that do this should also implement a [Compiler Browser Symbol Entry Point](#) which provides the IDE with the names of the custom symbol types.

For more information about the format of browser information, see [“Browser Reference” on page 573](#).

## Generating Debugging Data

To support debugging, compiler and linker plug-ins usually work in tandem to produce symbolic information in a format useful to a debugger. Typically, debuggers require information that assists in mapping offsets within executable code back to source code, and information about variables (whether local, global, or object data members) including their types and how to locate their contents at runtime. This information may be embedded in the final executable, or stored in a separate file, to be located and accessed by a debugger.

To determine if it should generate data to be used by a debugger, a compiler calls [CWIIsGeneratingDebugInfo](#). The result is true if debugging is enabled for the file currently being compiled (which is controlled by the bug icon in the project window). Otherwise, debugging is disabled.

When debugging is enabled, a compiler should store symbolic information required by the target linker. Usually, debugging information generated by a compiler is stored with the object code it

produces for a source file, which is stored in the project using [CWStoreObjectData](#).

When debugging is enabled, a linker should combine the symbolic information generated by compilers with any additional information about code and storage locations determined during the link phase, to produce final symbolic information for the target debugger.

Windows	For maximum interoperability, debugging information for Windows targets should conform to the standards established by Microsoft's CodeView debugger. Consult Microsoft for detailed information about this standard.
Mac OS	Debugging information for Mac OS targets should conform to the SYM and xSYM file formats established by Apple. SYM files are for 68K targets, and xSYM files are for PowerPC targets. Contact Apple for information about the SYM and xSYM file formats.
Embedded	Many embedded platform debuggers utilize the DWARF standard for symbolic information. This standard was produced by the Tool Interface Standards Committee.  Normally, debuggers which require symbolic information stored in separate files will require them to be located in the same directory as the target binary. plug-ins can locate this directory using <a href="#"><u>CWGetOutputFileDirectory</u></a> . In cases where the symbolics file and the target file do not reside in the same directory, plug-ins can specify the directory for the symbolic information using a preference setting. To retrieve preference settings, plug-ins call <a href="#"><u>CWGetNamedPreferences</u></a> .

## Additional Compiler and Linker Requests

There are additional optional requests that the IDE sends to compiler and linker plug-ins that request them. These requests indicate the start and end of various interesting events during a build process, and may be useful to plug-ins with special requirements. These requests include:

- `reqTargetLinkStarted` and `reqTargetLinkEnded`: the IDE sends these requests before and after linking a single target, respectively. Note that a single **Make** operation may build multiple targets.

## **Creating Compiler and Linker Plug-ins**

### *Additional Compiler and Linker Requests*

---

- `reqFileListBuildStarted` and `reqFileListBuildEnded`: the IDE sends these requests before and after compiling one or more files selected in the project file list, in response to the user selecting **Project > Compile**.
- `reqSubProjectBuildStarted` and `reqSubProjectBuildEnded`: the IDE sends these requests before and after compiling a subproject. A single **Make** operation may build multiple subprojects.
- `reqTargetBuildStarted` and `reqTargetBuildEnded`: the IDE sends these requests before and after compiling a single target, respectively. A single **Make** operation may build multiple targets.
- `reqProjectBuildStarted` and `reqProjectBuildEnded`: the IDE sends these requests before and after compiling a complete project, respectively. Plug-ins receive only one pair of project build messages during a **Make** operation.
- `reqTargetCompileStarted`: The IDE sends this request before starting a compilation.

For a typical single-target build operation, a plug-in receives only one pair of target related link and build requests. However, the IDE supports building multiple targets at once. In these cases, the plug-in will receive multiple pairs of requests during a single build operation. The same consideration applies to subprojects.

For a simple project containing two targets having three files each, a compiler plug-in might receive the following sequence of requests:

1. `reqInitialize`
2. `reqProjectBuildStarted`
3. `reqTargetBuildStarted`
4. `reqTargetCompileStarted`
5. `reqCompile`
6. `reqCompile`
7. `reqCompile`
8. `reqTargetCompileEnded`
9. `reqTerminate`
10. `reqInitialize`

```
11.reqTargetBuildEnded  
12.reqTargetBuildStarted  
13.reqCompile  
14.reqCompile  
15.reqCompile  
16.reqTerminate  
17.reqInitialize  
18.reqTargetBuildEnded  
19.reqProjectBuildEnded  
20.reqTerminate
```

Note the following:

- The plug-in is not guaranteed to receive `reqInitialize` and `reqTerminate` requests at any particular time. These requests are simply sent when a plug-in is loaded and unloaded, which can happen at various unpredictable times. In the example above, a plug-in might not receive the initial `reqInitialize` request, since most likely it has already been initialized.
- The IDE unloaded the compiler at steps 7 and 14 in order to free more memory for linking, and then reloaded it at steps 8 and 15, in order to send the `reqTargetBuildEnded` request.
- Target linking occurred between steps 7 and 8, and steps 14 and 15. Because the IDE unloads the compiler to perform linking, the compiler does not receive `reqTargetLinkStarted` and `reqTargetLinkEnded` requests.
- The IDE may unload and reload a compiler or linker between receipt of a “start” request and its matching “end” request. To take advantage of such requests, plug-ins typically must use target storage to save the results of the start request until the end request arrives.

In order to receive such events, which are not commonly required by most plug-ins, a plug-in must set the appropriate bits in its [DropInFlags](#). See [Compiler and Linker DropIn Flags](#) for more information.

# Managing Projects and Targets

This section discusses services used by compilers and linkers to obtain and change information about the current target, to manage global target data, and to manage object code and linker output.

The topics in this section are:

- [Adding a File to a Project](#)
- [Removing a File from a Project](#)
- [Storing Object, Resource, and Class Browser Data](#)
- [Getting Object Data and Resource Data](#)
- [Specifying File Dependencies](#)
- [Caching File Data](#)
- [Storing Linker Output](#)
- [Storing Post-Linker Output](#)
- [Getting Linker Output](#)
- [Getting Runtime Settings](#)

## Adding a File to a Project

To add a file to the project, a plug-in uses [CWAddProjectEntry](#). If the plug-in is a compiler, the IDE will initiate compilation of the newly-added file. If the plug-in is a pre-linker, linker, or post-linker, the plug-in must also add object data for the file by calling [CWStoreObjectData](#), since the IDE will not compile files added during a build operation.

[CWAddProjectEntry](#) is typically used by source-to-source compilers. For example, a compiler that translated Eiffel to C might examine source files written in Eiffel and produce '.c' files, and add them to the project using [CWAddProjectEntry](#).

Note that there is no need to remove older, out of date source files, or to determine whether a file has already been added to a project. In all cases, a compiler may simply call [CWAddProjectEntry](#) and the IDE will properly either add the file, or update it's dependency information based upon the time of modification of the file.

Source-to-source compilers which generate files that aren't necessarily part of a project should be sure to call [CWSetModDate](#) to assist the IDE in tracking changes to the file, and in ensuring tracking dependencies. This would typically occur for header files generated by a source-to-source compiler. Failure to call [CWSetModDate](#) can result in incorrect dependency tracking, and incorrectly updated builds.

---

**NOTE** It isn't sufficient to merely modify the OS-maintained time stamp on a generated output file. Plug-ins must call [CWSetModDate](#), otherwise the IDE will not notice changes to the file.

---

Usually, plug-ins will want to set the `isGenerated` flag when calling [CWAddProjectEntry](#) or [CWSetModDate](#). The exception to this is when a file is added to a project that existed outside the project previously, and which the plug-in cannot regenerate. This is a rare occurrence.

## Removing a File from a Project

To remove a file from a project, a plug-in uses [CWRemoveProjectEntry](#). CWRemoveProjectEntry also removes any object code associated with the file.

If the removed file was depended on by other files, the other files may be marked dirty. In such a case, the plug-in can rebuild the project. If it matters to the plug-in that a file was removed and the mod date consequently needs to be changed once the plug-in detects that a file was removed, the plug-in can call [CWSetModDate](#) to change the date of the other files.

## Storing Object, Resource, and Class Browser Data

To store object data, resource data, and class browser data for a file in the project, a plug-in fills in the fields of a [CWOBJECTDATA](#) structure and passes it to the IDE by calling [CWStoreObjectData](#).

The plug-in stores object data or resource data as a [CWMemHandle](#) in the `objectdata` field of a [CWOBJECTDATA](#) structure. Whether

the data stored using [CWStoreObjectData](#) contains object code or resources (or both) is up to a compiler and its associated linker.

Plug-ins store browser data in the `browsedata` field, also as a [CWMemHandle](#). The format of the data stored in the `browsedata` handle must match that described in [“Browser Reference” on page 573](#).

---

**TIP** To examine a plug-in’s browser data output for debugging purposes, use the **Dump internal browse information after compile** checkbox in the **Build Extras** settings panel.

---

When a plug-in calls [CWStoreObjectData](#), the IDE stores object, resource, and browser data for the file, and marks the file as “untouched.” It also commits all file dependencies established through [CWFindAndLoadFile](#) and through the dependencies array specified when calling [CWStoreObjectData](#).

## Getting Object Data and Resource Data

To get the object or resource data for an individual file in the project, a plug-in uses [CWLoadObjectData](#). [CWLoadObjectData](#) returns a [CWMemHandle](#) containing the object data that was compiled from a source code file, or the object data of a compiled or imported library file added directly to a project. Once a plug-in has finished using the object data for a file, it should call [CWFreeObjectData](#) to dispose the object data.

If a compiler stores its object code externally, the `objectdata` handle returned by a subsequent call to [CWLoadObjectData](#) will contain the contents of the file, which the IDE loads on behalf of the plug-in. The location of the external object file may be retrieved by calling [CWGetStoredObjectFileSpec](#).

To determine where to store object files initially, compilers should call [CWGetSuggestedObjectFileSpec](#), which returns the directory where the IDE stores data for the current target. Compilers should use this location for external object files, if possible, to facilitate movement and copying of projects and resolution of file name clashes by the IDE.

## Specifying File Dependencies

The IDE tracks dependencies among files in a project, to automate the rebuild process. By tracking dependencies, the IDE can determine which files must be recompiled when other files change. Plug-ins can inform the IDE of dependencies using one of two routines: [CWFindAndLoadFile](#) or [CWStoreObjectData](#).

[CWFindAndLoadFile](#) can be used to specify a single file dependency, and optionally load a file. Use the [CWFileInfo](#) parameter to specify dependencies and other options, such as whether to load a file or just find it. Most plug-ins use [CWFindAndLoadFile](#) to establish dependencies, after extracting the name of a file to load from source text. Usually, included files are specified by name, or partial path, and plug-ins benefit from the project access path search performed by [CWFindAndLoadFile](#).

[CWStoreObjectData](#) also allows a compiler to specify dependencies when adding object code to a project. Unlike [CWFindAndLoadFile](#), [CWStoreObjectData](#) can specify multiple dependencies at one time. [CWStoreObjectData](#) is most often used to store dependency information when the files depended upon are located using unconventional mechanisms (and so calling [CWFindAndLoadFile](#) may be impossible or undesirable).

Dependencies are specified using the dependencyCount and dependencies fields of a [CWOBJECTDATA](#) structure. The dependencies array lists all files depended upon by the file whose object data is being stored.

Dependencies are specified using a [CWDependencyInfo](#) structure, and can be specified by file index or by file specification. The fields of a [CWDependencyInfo](#) structure specify the dependency type (see [CWDependencyType](#)), and the method the IDE uses to find the files.

Plug-ins for languages with relatively simple dependency relationships typically call [CWFindAndLoadFile](#) to load files and simultaneously notify the IDE of dependencies. Plug-ins that adapt command line tools (which cannot call [CWFindAndLoadFile](#)), and Plug-ins for languages whose dependency relationships are not explicitly specified in source directives (making it inconvenient to call [CWFindAndLoadFile](#)), typically call [CWStoreObjectData](#).

## Caching File Data

The IDE provides a mechanism for caching precompiled header data. Caching such data potentially improves compilation performance, since headers chosen for precompilation are typically large, and may be used by many files in a project. Caching improves performance by reducing the need to load data from disk.

Before caching precompiled header data, a plug-in must first check to see if precompiled header caching is enabled, by calling [`CWIsCachingPrecompiledHeaders`](#). If the result is true, the plug-in may call [`CWCachePrecompiledHeader`](#) to store precompiled interface data in the IDE's RAM cache. The next time a plug-in calls [`CWFindAndLoadFile`](#) using the file specification of a cached precompiled header, the IDE will return the cached data.

Note that the IDE does not cache precompiled headers automatically. Plug-ins must explicitly indicate which precompiled headers to cache by calling [`CWCachePrecompiledHeader`](#).

Also note that [`CWFindAndLoadFile`](#) will not load the data for a precompiled header file if it no longer resides in the cache. If the data for a precompiled header has been previously cached, and remains in the cache, [`CWFindAndLoadFile`](#) will return the cached data. Otherwise, [`CWFindAndLoadFile`](#) will return NULL for the file data. In this case, the plug-in must use the returned file specification to reload the precompiled header data (and cache it again, if desired).

---

**NOTE**

Because the IDE's precompiled header cache is currently limited to two files, plug-ins should selectively cache files, rather than caching all files. Prefix files (implicitly included in all project files) are good candidates for caching.

---

## Managing Target Storage

Plug-ins may store arbitrary data on a per-target basis using [`CWGetTargetStorage`](#) and [`CWSetTargetStorage`](#). The IDE retains target storage in memory for the duration of one invocation of the IDE, and keeps separate data for each target that a plug-in operates upon. In effect, target storage is global storage reserved for

use by plug-ins, but maintained by the IDE, and only accessible through the plug-in API.

Target storage is useful for storing data which a plug-in desires to maintain globally, even when the plug-in is unloaded and reloaded by the IDE. Target storage is also useful in cases where global state reinitialization is an expensive operation. Since the IDE can and does unload tools fairly often, this can be an important performance optimization.

A plug-in should allocate and initialize its target data when the IDE sends a [reqTargetLoaded](#) request. A plug-in should deallocate or clean up its target data when it receives a [reqTargetUnloaded](#) request from the IDE.

To allocate and deallocate memory for target storage, a plug-in calls [CWAllocateMemory](#), and [CWFreeMemory](#) respectively. To ensure that target storage is retained between requests, instead of being automatically disposed by the IDE, plug-ins should call [CWAllocateMemory](#) with `ispermanent` set to true. [Listing 6.6](#) illustrates this.

#### **Listing 6.6 Handling reqTargetLoaded and reqTargetUnloaded requests**

```
CWPLUGIN_ENTRY (main) (CWPluginContext context)
{
    CWResult      result = cwNoErr;
    CWResult      error = cwNoErr;
    long          request;

    /* Get the plug-in request */
    result = CWGetPluginRequest(context, &request);

    if (result != cwNoErr)
        return result;

    /* dispatch on request */
    switch (request)
    {
        case reqInitialize:
            /* tool has just been loaded into memory */
            break;
    }
}
```

## **Creating Compiler and Linker Plug-ins**

### *Managing Target Storage*

---

```
case reqTerminate:
    /* tool is about to be unloaded from memory */
    break;

case reqTargetLoaded:
    /* a new target has been loaded; */
    /* allocate and set target storage */
    error = DoTargetLoaded (context);
    break;

case reqTargetUnloaded:
    /* current target is about to be unloaded; */
    /* dispose and clear target storage */
    error = DoTargetUnloaded (context);
    break;

/* ... handle other requests here ... */
/* requests that require target storage */
/* should call CWGetTargetStorage */

default:
    error = cwErrRequestFailed;
    break;
}

// Tell the IDE the request has been completed
result = CWDonePluginRequest(context, error);

// return result code
return (result);
}

typedef struct pluginStuff
{
    /* this structure holds global information the plug-in wants */
    /* to retain even when the plug-in is loaded and unloaded. */
    longint          A;
    void            * Buffer;
} pluginStuff, * pluginStuffPtr;

CWResult DoTargetLoaded (CWPluginContext context)
{
```

```
void           * MemPtr;
pluginStuffPtr myPluginStuff;
CWResult      error = cwNoErr;
CWResult      dontCare = cwNoErr;

error = CWAllocateMemory (context, sizeof(pluginStuff),
                         true, &MemPtr);
if (error == cwNoErr)
{
    myPluginStuff = pluginStuffPtr (MemPtr);

    /* perform any initialization of global data here */
    myPluginStuff->A = 1;
    error = CWAllocateMemory (context, kBufferSize,
                             true, &myPluginStuff->Buffer);

    if (error == cwNoErr)
    {
        /* save target storage */
        error = CWSetTargetStorage (context, MemPtr);
    }
    else
    {
        /* clean up partial allocations */
        dontCare = CWFreeMemory (context, MemPtr);
    }
}
return error;
}

CWResult DoTargetUnloaded (CWPluginContext context)
{
    void           * MemPtr;
    pluginStuffPtr myPluginStuff;
    CWResult      error = cwNoErr;
    CWResult      error2 = cwNoErr;

    error = CWGetTargetStorage (context, &MemPtr);
    if (Err == cwNoErr)
    {
        myPluginStuff = pluginStuffPtr (MemPtr);
```

```
/* clean up any global storage allocations here */
if (myPluginStuff->Buffer != NULL)
    error = CWFreeMemory (context, myPluginStuff->Buffer);

/* clear target storage (even if error) */
error2 = CWSetTargetStorage (context, NULL);
if (error == cwNoErr)
    error = error2;
}
return error;
}
```

---

[CWGetTargetStorage](#) and [CWSetTargetStorage](#) are only available to compiler and linker plug-ins. In addition, plug-ins may only utilize target storage if they set the appropriate dropin flags ([kCompUsesTargetStorage](#) and [linkerUsesTargetStorage](#)).

Although plug-ins normally dispose target storage in response to a [reqTargetUnloaded](#) request, in rare cases plug-ins may need to dispose target storage before the IDE sends this request. This might be the case if the target storage allocation is large, the plug-in has finished using its target storage, and the plug-in expects to make additional large allocations for other purposes. In such cases, a plug-in is free to dispose its target storage early. However, to ensure that all code that uses the target storage is properly informed of the disposal, the plug-in should call [CWSetTargetStorage](#) with a NULL target storage pointer, immediately after disposing its target storage.

## Storing Linker Output

Typically, a linker saves its final output to a file. However, the linker can also append its output to a file. Use [CWPutResourceFile](#) to prompt the user for a file specification of a file to append linker output to. This is most useful when merging binary resources and final linker output into a compiled application on Mac OS.

A plug-in should record information about the output file it generates so that it may be returned to the IDE when handling a [reqTargetInfo](#) request. Since linking and returning target information are separate requests, a plug-in must store information

about the linker output globally. This can be done by storing linker output information in target storage, allocated and freed in response to [reqTargetLoaded](#) and [reqTargetUnloaded](#) requests. See [Managing Target Storage](#) for more information.

Often, however, much of this information is stored in settings panel data for the linker, and can simply be loaded by calling [CWGetNamedPreferences](#). For more information see [“Providing Target Information” on page 216](#).

## Storing Post-Linker Output

The IDE places no requirements on where a post linker stores its output. Also, unlike a regular linker, a post linker is not required to act on a [reqTargetInfo](#) request. For more information on [reqTargetInfo](#) see [“Providing Target Information” on page 216](#).

## Getting Linker Output

To get information about the linker’s output, a plug-in calls [CWGetTargetInfo](#). The information returned specifies the location of the linked executable or library, the debugging symbolics file, and any application necessary to support running or debugging the final output. It also specifies the final binary type, the target OS and CPU, and other information needed when debugging.

---

**NOTE** The information returned by [CWGetTargetInfo](#) may reflect changes made to the target info by a post linker as well as the linker.

---

## Getting Runtime Settings

When a linker or post-linker utilizes a debug or run helper application, the IDE delegates responsibility for launching the helper application to the plug-in. Often, the plug-in will wish to pass certain information to the helper process, and possibly it may set up the runtime environment for the process prior to launching it.

The IDE provides a **Runtime Settings** preference panel that allows the IDE user to configure various aspects of the runtime

environment for helper applications. In addition, it provides IDE calls that allow plug-ins to retrieve this information.

To determine the desired working directory for helper applications, plug-ins call [CWGetWorkingDirectory](#), and switch the operating system or command line shell's current default directory to this location.

In the event the helper application is a command line tool, the plug-in may also call [CWGetCommandLineArgs](#) to obtain any command line switches that the user configures in project preferences. These are intended to be appended to any command line passed to the command line tool (which may include additional switches added by the linker plug-in).

Plug-ins can obtain any environment variables configured for the helper application by calling [CWGetEnvironmentVariable](#) and [CWGetEnvironmentVariableCount](#). [CWGetEnvironmentVariableCount](#) returns the number of configured environment variables, which can be enumerated using [CWGetEnvironmentVariable](#). Each environment variable is simply a variable name and an associated value, both of which are represented as text. Again, the plug-in is responsible for establishing these settings prior to launching the helper application.

These routines primarily apply to Windows. Mac OS has no command line, and therefore some options, such as command line arguments, usually do not apply (although the Metrowerks Java tools for Mac OS utilize settings from the **Runtime Settings** panel).

# Compiler and Linker Plug-in Reference

---

This chapter describes the plug-in API services available to compilers, linkers, pre-linkers, and post-linkers.

## Overview

This chapter covers the following topics:

- [Routines for Compiler and Linker Plug-ins](#)
- [User Routines for Compiler and Linker Plug-ins](#)
- [Data Structures for Compiler and Linker Plug-ins](#)
- [Constants for Compiler and Linker Plug-ins](#)
- [Result Codes for Compiler and Linker Plug-ins](#)

## Routines for Compiler and Linker Plug-ins

This section lists the routines a compiler, linker, pre-linker, or post-linker may call.

### Alphabetical Routine Index

This section lists all compiler and linker routines alphabetically.

- [CWCachePrecompiledHeader](#)
- [CWDisplayLines](#)
- [CWFreeObjectData](#)
- [CWGetBrowseOptions](#)
- [CWGetBuildSequenceNumber](#)

- [CWGetCommandLineArgs](#)
- [CWGetEnvironmentVariable](#)
- [CWGetEnvironmentVariableCount](#)
- [CWGetMainFileID](#)
- [CWGetMainFileName](#)
- [CWGetMainFileSpec](#)
- [CWGetMainFileText](#)
- [CWGetModifiedFiles](#)
- [CWGetPrecompiledHeaderSpec](#)
- [CWGetResourceFile](#)
- [CWGetStoredObjectFileSpec](#)
- [CWGetSuggestedObjectFileSpec](#)
- [CWGetTargetInfo](#)
- [CWGetTargetStorage](#)
- [CWGetWorkingDirectory](#)
- [CWIIsAutoPrecompiling](#)
- [CWIIsCachingPrecompiledHeaders](#)
- [CWIIsGeneratingDebugInfo](#)
- [CWIIsPrecompiling](#)
- [CWIIsPreprocessing](#)
- [CWLoadObjectData](#)
- [CWPutResourceFile](#)
- [CWSetTargetInfo](#)
- [CWSetTargetStorage](#)
- [CWStoreObjectData](#)

## Functional Routine Index

This section lists all routines grouped by function.

### Current File Information

- [CWGetMainFileID](#)
- [CWGetMainFileName](#)

- [CWGetMainFileSpec](#)
- [CWGetMainFileText](#)

### **Request Handling**

- [CWIIsAutoPrecompiling](#)
- [CWIIsCachingPrecompiledHeaders](#)
- [CWIIsGeneratingDebugInfo](#)
- [CWIIsPrecompiling](#)
- [CWIIsPreprocessing](#)
- [CWGetBrowseOptions](#)
- [CWGetBuildSequenceNumber](#)

### **Managing Object Data**

- [CWFreeObjectData](#)
- [CWLoadObjectData](#)
- [CWStoreObjectData](#)
- [CWGetStoredObjectFileSpec](#)
- [CWGetSuggestedObjectFileSpec](#)

### **Managing Precompiled Headers**

- [CWCachePrecompiledHeader](#)
- [CWGetPrecompiledHeaderSpec](#)

### **Managing Target Information**

- [CWGetTargetInfo](#)
- [CWSetTargetInfo](#)
- [CWGetModifiedFiles](#)

### **Managing Target Storage**

- [CWGetTargetStorage](#)
- [CWSetTargetStorage](#)

### **Target Runtime Settings**

- [CWGetCommandLineArgs](#)

- [CWGetEnvironmentVariable](#)
- [CWGetEnvironmentVariableCount](#)
- [CWGetWorkingDirectory](#)

### User Interaction

- [CWDisplayLines](#)
- [CWGetResourceFile](#)
- [CWPutResourceFile](#)

## CWWCachePrecompiledHeader

Description	Stores precompiled header data in a RAM cache for quicker access.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWWCachePrecompiledHeader( <a href="#">CWPluginContext</a> context, const <a href="#">CWFileSpec</a> * filespec, <a href="#">CWMemHandle</a> pchhandle);
Parameters	Parameters for this function are:  context                 Private, opaque IDE state. filespec                 Specifies the location of the precompiled header file corresponding to the header data specified in pchhandle. pchhandle                 Contains the precompiled header data.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Plug-ins store precompiled header data in pchhandle, and the file specification for the corresponding precompiled header file in filespec. After successfully caching a header, subsequent requests to load the file made using <a href="#">CWFindAndLoadFile</a> will return the cached data.
<b>NOTE</b>	Currently, only two precompiled headers may be stored in the precompiled header cache.

---

Before calling CWCachePrecompiledHeader, a plug-in should call [CWIsCachingPrecompiledHeaders](#) to ensure that the IDE has sufficient memory to cache precompiled headers. If [CWIsCachingPrecompiledHeaders](#) returns true, the plug-in may call CWCachePrecompiledHeader. A true result does not guarantee that CWCachePrecompiledHeader will succeed. Plugins should still check for successful completion.

See Also

[“CWMemHandle” on page 166](#)

[“CWIsCachingPrecompiledHeaders” on page 260](#)

## CWDisplayLines

Description Reports progress information to the IDE for display during the current compile.

Prototype

```
#include <DropInCompilerLinker.h>
CW_CALLBACK CWDisplayLines(
    CWPluginContext context,
    long nlines);
```

Parameters Parameters for this function are:

context	Private, opaque IDE state.
nlines	Specifies the number of line processed so far by the plug-in.

Return An error code listed in [“Result Codes for Compiler and Linker Plug-ins” on page 328](#) or [“Result Codes for Plug-ins” on page 193](#)

Remarks Call this routine regularly during a compile operation to update the IDE's progress window. plug-ins should call this routine often enough to provide good feedback, but not so often as to adversely affect performance. A good rule of thumb is to call CWDisplayLines once per 50-100 lines of source code, or several times per second.

---

**WARNING!**

Be careful to ensure that whatever logic is used to determine when to call CWDisplayLines is fast. Otherwise, simply determining when to call this routine can affect performance.

---

See Also

[“CWShowStatus” on page 145](#)

## CWFreeObjectData

Description	Disposes memory allocated by <a href="#">CWLoadObjectData</a> .
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWFreeObjectData( <a href="#">CWPluginContext</a> context, long whichfile, <a href="#">CWMemHandle</a> objectdata);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. whichfile       Specifies the file by index, in target link order, corresponding to the object data to release. objectdata      Contains the object data to be disposed, returned by a previous call to <a href="#">CWLoadObjectData</a> .
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Call CWFreeObjectData to free object data previously loaded by <a href="#">CWLoadObjectData</a> . Be careful to pass the index in whichfile of the file corresponding to the data passed in objectdata.
See Also	<a href="#">“CWLoadObjectData” on page 263</a>

## CWGetBrowseOptions

Description	Tells a plug-in what types of browser symbols to generate.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetBrowseOptions( <a href="#">CWPluginContext</a> context, <a href="#">CWBrowseOptions</a> * browseOptions);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. browseOptions   Returns a <a href="#">CWBrowseOptions</a> structure specifying the types of symbols the plug-in should generate.

Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	The IDE expects all compiler plug-ins that support generation of browser information to generate information for global variables and routines. A compiler plug-in should call <code>CWGetBrowseOptions</code> to determine which additional types of browser data should be generated.
<b>NOTE</b>	In practice, the IDE currently always enables all browser symbol types. However, plug-ins should still respect the flags returned by this call; browser operation may change in the future.
See Also	<a href="#">“CWBrowseOptions” on page 274</a> <a href="#">“reqCompile” on page 313</a>

## CWGetBuildSequenceNumber

Description	Returns a unique ID for the current <b>Make</b> operation.				
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; CW_CALLBACK CWGetBuildSequenceNumber(     <a href="#">CWPluginContext</a> context,     long* sequenceNumber);</pre>				
Parameters	<p>Parameters for this function are:</p> <table border="0"> <tr> <td style="vertical-align: top;">context</td> <td>Private, opaque IDE state.</td> </tr> <tr> <td style="vertical-align: top;">sequenceNumber</td> <td>Returns the unique IDE of the current <b>Make</b> operation.</td> </tr> </table>	context	Private, opaque IDE state.	sequenceNumber	Returns the unique IDE of the current <b>Make</b> operation.
context	Private, opaque IDE state.				
sequenceNumber	Returns the unique IDE of the current <b>Make</b> operation.				
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>				
Remarks	The IDE assigns each <b>Make</b> operation a unique ID number. By retrieving this number with <code>CWGetBuildSequenceNumber</code> and storing it in persistent memory allocated with <code>CWAllocateMemory</code> , a plug-in may determine if the current request and previous requests apply to the same <b>Make</b> operation.				
See Also	<a href="#">“CWGetTargetStorage” on page 256</a>				

[“CWSetTargetStorage” on page 266](#)

[“CWAllocateMemory” on page 106](#)

## **CWGetCommandLineArgs**

Description	Returns the text entered by the user in the <b>Program Arguments</b> text box of the <b>Runtime Settings</b> panel.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetCommandLineArgs( <a href="#">CWPluginContext</a> context, const char** commandLineArgs);
Parameters	Parameters for this function are:  context                          Private, opaque IDE state. commandLineArgs                 Returns a pointer to a constant C string containing the command line parameters entered in the <b>Program Arguments</b> field of the <b>Runtime Settings</b> panel.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	This routine returns any command line options that the user has specified in the <b>Program Arguments</b> field of the <b>Runtime Settings</b> panel. These command line arguments apply to the execution of the final target executable.
Mac OS	This routine is supported on Mac OS hosted IDEs, but only returns useful information when the current linker enables the <b>Runtime Settings</b> panel. Currently, this happens for the Java linker.
See Also	<a href="#">“CWGetEnvironmentVariable” on page 244</a> <a href="#">“CWGetEnvironmentVariableCount” on page 246</a> <a href="#">“CWGetWorkingDirectory” on page 257</a>

## **CWGetEnvironmentVariable**

Description	Returns the value of an environment variable as configured in the <b>Environment Settings</b> list box of the <b>Runtime Settings</b> panel.
-------------	--

Prototype    `#include <DropInCompilerLinker.h>`  
`CW_CALLBACK CWGetEnvironmentVariable(`  
`CWPluginContext context,`  
`long index,`  
`const char** name,`  
`const char** value);`

Parameters	Parameters for this function are:
	context        Private, opaque IDE state.
	index          Specifies which environment variable to return, by index.
	name           Returns a pointer to a constant C string containing the name of the runtime environment variable specified by index.
	value          Returns a pointer to a constant C string containing the value of the runtime environment variable specified by index.

Return    An error code listed in [“Result Codes for Compiler and Linker Plug-ins” on page 328](#) or [“Result Codes for Plug-ins” on page 193](#)

Remarks    `CWGetEnvironmentVariable` retruns the name and value of environment variables as established in the **Environment Settings** list box of the **Runtime Settings** panel.

The values returned control runtime settings established prior to launching the target executable, not to the current IDE environment variable settings. `CWGetEnvironmentVariable` is most commonly used by linker plug-ins, to set up the proper runtime environment for command targets prior to execution.

To obtain all runtime environment variable settings, call `CWGetEnvironmentVariable` repeatedly with an `index` value ranging from 1 to the count returned by [`CWGetEnvironmentVariableCount`](#).

Mac OS    This routine is supported on Mac OS hosted IDEs, but only returns useful information when the current linker enables the **Runtime Settings** panel. Currently, this happens for the Java linker.

See Also    [“CWGetCommandLineArgs” on page 244](#)

[“CWGetEnvironmentVariableCount” on page 246](#)

[“CWGetWorkingDirectory” on page 257](#)

## CWGetEnvironmentVariableCount

Description	Returns the number of runtime environment variables configured in the <b>Environment Settings</b> list box of the <b>Runtime Settings</b> panel.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetEnvironmentVariableCount( <a href="#">CWPluginContext</a> context, long* count);
Parameters	Parameters for this function are:  context      Private, opaque IDE state. count         Returns the count of environment variable settings established in the <b>Environment Settings</b> list box of the <b>Runtime Settings</b> panel.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Plug-ins use CWGetEnvironmentVariableCount to determine the number of environment variable settings the user has specified for the current target in its <b>Runtime Settings</b> panel. This is useful when iterating over all runtime environment variable settings, usually in preparation for launching a final executable. Typically, linkers obtain all environment variables, and pass them to a helper application, which sets up the runtime environment for the target application prior to launch.
Mac OS	This routine is supported on Mac OS hosted IDEs, but only returns useful information when the current linker enables the <b>Runtime Settings</b> panel. Currently, this happens for the Java linker.
See Also	<a href="#">“CWGetCommandLineArgs” on page 244</a> <a href="#">“CWGetEnvironmentVariable” on page 244</a> <a href="#">“CWGetWorkingDirectory” on page 257</a>

## CWGetMainFileID

Description	Returns the ID of the file currently being processed in the active project target.
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; CW_CALLBACK CWGetMainFileID(     <a href="#">CWPluginContext</a> context,     short* fileID);</pre>
Parameters	Parameters for this function are:  context      Private, opaque IDE state. fileID        Returns the file ID of the file to process.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	The IDE assigns each file in a project a unique ID. Plug-ins use these IDs when emitting browser information to specify source files.  A compiler plug-in calls <code>CWGetMainFileID</code> in response to a <a href="#">reqCompile</a> request, to determine the browser ID of the file currently being compiled. A compiler then emits this ID in browser information to specify the current file as the source file in which a symbol originates.  File IDs may also be useful for tracking project files concisely. File IDs persist between instances of starting and terminating the IDE and opening and closing a project. To determine the file ID of any project file, or to determine which file has a given ID, use <a href="#">CWGetFileInfo</a> .
See Also	<a href="#">“CWGetMainFileName” on page 248</a> <a href="#">“CWGetMainFileSpec” on page 249</a> <a href="#">“CWGetMainFileText” on page 249</a> <a href="#">“CWGetFileInfo” on page 120</a> <a href="#">“CWProjectFileInfo” on page 174</a> <a href="#">“CWStoreObjectData” on page 267</a>

[“reqCompile” on page 313](#)

## CWGetMainFileName

Description	Returns the index of the file currently being processed.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetMainFileName( <a href="#">CWPluginContext</a> context, long*               fileNumber);
Parameters	Parameters for this function are:  context              Private, opaque IDE state. fileNumber           Returns the index of the file to process, in target link order.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	In response to a <a href="#">reqCompile</a> , <a href="#">reqCompDisassemble</a> , or <a href="#">reqCheckSyntax</a> request, a plug-in determines the index of the file being compiled using CWGetMainFileName. The value returned by CWGetMainFileName may be used to call routines such as <a href="#">CWLoadObjectData</a> and <a href="#">CWStoreObjectData</a> .
See Also	<a href="#">“CWGetMainFileID” on page 247</a> <a href="#">“CWGetMainFileSpec” on page 249</a> <a href="#">“CWGetMainFileText” on page 249</a> <a href="#">“CWLoadObjectData” on page 263</a> <a href="#">“CWStoreObjectData” on page 267</a> <a href="#">“reqCompile” on page 313</a> <a href="#">“reqCompDisassemble” on page 313</a> <a href="#">“reqCheckSyntax” on page 312</a>

## CWGetMainFileSpec

Description	Returns the file specification of the file currently being processed.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetMainFileSpec( <a href="#">CWPluginContext</a> context, <a href="#">CWFfileSpec</a> * fileSpec);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. fileSpec        Returns the file specification of the file currently being processed.
Return	An error code listed in “ <a href="#">Result Codes for Compiler and Linker Plug-ins</a> ” on page 328 or “ <a href="#">Result Codes for Plug-ins</a> ” on page 193
Remarks	During a <a href="#">reqCompile</a> , <a href="#">reqCompDisassemble</a> , or <a href="#">reqCheckSyntax</a> request, a plug-in determines the location of the file to operate on using CWGetMainFileSpec. The value returned by CWGetMainFileSpec may be used to retrieve information about the file, or the file’s contents using <a href="#">CWGetFileText</a> . To load the text of the file currently being compiled, plug-ins may instead call <a href="#">CWGetMainFileText</a> .
See Also	<a href="#">“CWGetMainFileID” on page 247</a> <a href="#">“CWGetMainFileNumber” on page 248</a> <a href="#">“CWGetMainFileText” on page 249</a> <a href="#">“reqCompile” on page 313</a> <a href="#">“reqCompDisassemble” on page 313</a> <a href="#">“reqCheckSyntax” on page 312</a>

## CWGetMainFileText

Description	Loads the contents of the file currently being processed.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetMainFileText( <a href="#">CWPluginContext</a> context,

```
const char**      text,
long*           textLength);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	text          Returns a pointer to memory containing the file's contents.
	textLength    Returns the size in bytes of the file's contents.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	In response to a <a href="#">reqCompile</a> , <a href="#">reqCompDisassemble</a> , or <a href="#">reqCheckSyntax</a> request, a plug-in retrieves the contents of the file to operate on using CWGetMainFileText.
<b>WARNING!</b>	Plug-ins should <i>not</i> release the text pointer returned by CWGetMainFileText.

<b>NOTE</b>	The file text returned by CWGetFileText is declared to be constant. plug-ins should not modify the text data returned in text.
-------------	--

See Also	<a href="#">“CWGetMainFileID” on page 247</a> <a href="#">“CWGetMainFileName” on page 248</a> <a href="#">“CWGetMainFileSpec” on page 249</a> <a href="#">“reqCompile” on page 313</a> <a href="#">“reqCompDisassemble” on page 313</a> <a href="#">“reqCheckSyntax” on page 312</a>
----------	---

## CWGetModifiedFiles

Description	Returns a list of source files that have been modified since the last time they were compiled.
Prototype	#include "DropinCompilerLinker.h"

```
CW_CALLBACK CWGetModifiedFiles(
    CWPluginContext context,
    long* modifiedFileCount,
    const long** modifiedFiles);
```

Parameters	Parameters for this function are:
	context                      Private, opaque IDE state.
	modifiedFileCount            Returns the count of modified files.
	modifiedFiles                Returns an array listing the modified files by index.
Remarks	This routine is provided mainly for Java, but may be useful for other languages. It returns a list of modified or “touched” files, which require recompilation.  The list returned is a constant array of long integer indexes which refer to other files in the current target by index. Typically, plug-ins use indexes returned in modifiedFiles to call <a href="#">CWGetFileInfo</a> .  The IDE allocates modifiedFiles on the plug-in’s behalf; the plug-in should not dispose this data (it will be disposed by the IDE at the end of the pending compile).
NOTE	The list of files returned is computed only once during a plug-in request; subsequent calls to CWGetModifiedFiles during the same request return the same list. The list of files does not change even if files are added to the project, or existing files are “touched.”
See Also	<a href="#">“CWGetFileInfo” on page 120</a>

## **CWGetPrecompiledHeaderSpec**

Description	Returns the recommended file specification for saving a precompiled header.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetPrecompiledHeaderSpec( <a href="#">CWPluginContext</a> context, <a href="#">CWFfileSpec</a> * pchspec,

## Compiler and Linker Plug-in Reference

### CWGetResourceFile

---

```
const char* target);
```

Parameters	Parameters for this function are:
	context Private, opaque IDE state.
	pchspec Returns a file specification for a precompiled header file.
	target Specifies the desired name of the precompiled header file.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Plug-ins call CWGetPrecompiledHeaderSpec to determine where to save a precompiled header file. If target is NULL, the IDE prompts the user for a file specification and returns it in pchspec. Otherwise, the IDE creates a file specification using target, and returns it in pchspec.  Plug-ins normally call CWGetPrecompiledHeaderSpec in response to a <a href="#">reqCompile</a> request for which one of <a href="#">CWIsPrecompiling</a> or <a href="#">CWIsAutoPrecompiling</a> returns true.
See Also	<a href="#">“CWIsPrecompiling” on page 261</a> <a href="#">“CWIsAutoPrecompiling” on page 258</a> <a href="#">“Precompiling” on page 212</a>

## CWGetResourceFile

Description Prompts the user to select an existing file containing resources.

```
#include <DropInCompilerLinker.h>
CW_CALLBACK CWGetResourceFile(
    CWPluginContext context,
    CWFleSpec* filespec);
```

Parameters	Parameters for this function are:
	context Private, opaque IDE state.
	filespec Returns the file specification of the resource file chosen by the user.

Return	An error code listed in “ <a href="#">Result Codes for Compiler and Linker Plug-ins</a> ” on page 328 or “ <a href="#">Result Codes for Plug-ins</a> ” on page 193
Remarks	<p><code>CWGetResourceFile</code> displays a standard file selection dialog box containing resource files and waits for the user to select a file. If the user cancels, <code>CWGetResourceFile</code> returns <a href="#"><code>cwErrUserCanceled</code></a>.</p> <p><code>CWGetResourceFile</code> is currently used by the Mac OS 68K and PowerPC linkers when compiling code resources for targets with both the <b>Display Dialog</b> and <b>Merge to File</b> checkboxes set. Other resource compilers may find it useful as well.</p>
<b>WARNING!</b>	Due to threading issues in all versions of CodeWarrior through Pro 5, linkers should not call this routine unless they support handling the <a href="#"><code>reqTargetInfo</code></a> request reentrantly, in which case they must also set the <a href="#"><code>linkerGetTargetInfoThreadSafe</code></a> dropin flag.

See Also      [“CWPutResourceFile” on page 264](#)

## CWGetStoredObjectFileSpec

Description	Returns the file specification most recently stored with a file in a call to <a href="#"><code>CWStoreObjectData</code></a> .						
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; CW_CALLBACK CWGetStoredObjectFileSpec(     <a href="#"><code>CWPPluginContext</code></a> context,     long whichfile,     <a href="#"><code>CWFfileSpec</code></a>* fileSpec);</pre>						
Parameters	Parameters for this function are:						
	<table><tr><td>context</td><td>Private, opaque IDE state.</td></tr><tr><td>whichfile</td><td>Specifies the file for which to obtain a <a href="#"><code>CWFfileSpec</code></a>, by index in target link order.</td></tr><tr><td>fileSpec</td><td>Returns the <a href="#"><code>CWFfileSpec</code></a> associated with a file, if any was specified when a plug-in last called <a href="#"><code>CWStoreObjectData</code></a>.</td></tr></table>	context	Private, opaque IDE state.	whichfile	Specifies the file for which to obtain a <a href="#"><code>CWFfileSpec</code></a> , by index in target link order.	fileSpec	Returns the <a href="#"><code>CWFfileSpec</code></a> associated with a file, if any was specified when a plug-in last called <a href="#"><code>CWStoreObjectData</code></a> .
context	Private, opaque IDE state.						
whichfile	Specifies the file for which to obtain a <a href="#"><code>CWFfileSpec</code></a> , by index in target link order.						
fileSpec	Returns the <a href="#"><code>CWFfileSpec</code></a> associated with a file, if any was specified when a plug-in last called <a href="#"><code>CWStoreObjectData</code></a> .						
Return	An error code listed in “ <a href="#">Result Codes for Compiler and Linker Plug-ins</a> ” on page 328 or “ <a href="#">Result Codes for Plug-ins</a> ” on page 193						

## Compiler and Linker Plug-in Reference

### CWGetSuggestedObjectFileSpec

---

**Remarks** CWGetStoredObjectFileSpec returns the [CWFfileSpec](#) for any externally stored object data file associated with a file in the current project target. whichfile specifies the file to get the external object file specification for, by index in target link order. The file specification returned is the one most recently stored by a plug-in when calling [CWStoreObjectData](#).

If the most recent call to [CWStoreObjectData](#) specified no file specification, CWGetStoredObjectFileSpec will return the error [cwErrObjectFileNotStored](#). The IDE determines whether a file specification has been associated with the file by examining the objectdata field of the [CWObjectData](#) structure. If objectdata is NULL, then the IDE assumes the objectfile file specification is to be ignored.

**See Also** [“CWStoreObjectData” on page 267](#)

[“CWObjectData” on page 281](#)

[“CWFfileSpec” on page 162](#)

[“cwErrObjectFileNotStored” on page 328](#)

## CWGetSuggestedObjectFileSpec

**Description** Returns the IDE's preferred location for externally stored object files.

**Prototype**

```
#include <DropInCompilerLinker.h>
CW_CALLBACK CWGetSuggestedObjectFileSpec(
    CWPluginContext context,
    long whichfile,
    CWFfileSpec* fileSpec);
```

**Parameters** Parameters for this function are:

context Private, opaque IDE state.

whichfile Specifies the file for which to return a suggested file specification.

fileSpec Returns the location of the folder in which the IDE wants external object files stored.

Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Plug-ins that need to store object data in files outside of the project should call <code>CWGetSuggestedObjectFileSpec</code> to determine where to store the object code files. This ensures that the IDE can properly manage the object code, rebuilding it when missing, loading it when the plug-in calls <a href="#">CWLoadObjectData</a> , and removing it prior to recompilation, or when the user selects <b>Project &gt; Remove Object Code</b> .
	The file specification returned specifies the location of the <i>directory</i> in which the plug-in should store object files, not the name of an object code file. The plug-in may use any desired file name within this folder.
	<code>CWGetSuggestedObjectFileSpec</code> is useful when implementing command line tool wrappers. Compiler plug-ins typically call <code>CWGetSuggestedObjectFileSpec</code> to determine where to place output files, pass this directory to command line tools via the command line, and specify this location in the <code>objectfile</code> field when calling <a href="#">CWStoreObjectData</a> . Linkers can determine the location and name of the object code file later, by calling <a href="#">CWGetStoredObjectFileSpec</a> .
See Also	<a href="#">“CWLoadObjectData” on page 263</a> <a href="#">“CWStoreObjectData” on page 267</a> <a href="#">“CWGetStoredObjectFileSpec” on page 253</a>

## CWGetTargetInfo

Description	Returns information about the current target.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetTargetInfo( <a href="#">CWPluginContext</a> context, <a href="#">CWTTargetInfo</a> * targetInfo);
Parameters	Parameters for this function are:  context                  Private, opaque IDE state. targetInfo               Returns information about the current target.

Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	CWGetTargetInfo returns information about the current link target, including target processor and OS, the final binary type, and target debugging information. Most of this information is determined by the linker currently selected in the <b>Target Settings</b> panel. The linker returns this information when handling the <a href="#">reqTargetInfo</a> request, by calling <a href="#">CWSetTargetInfo</a> . The IDE retains the returned information with the current target.
<b>NOTE</b>	Post-linkers can also override target information.
See Also	<a href="#">“CWTargetInfo” on page 283</a> <a href="#">“CWSetTargetInfo” on page 265</a> <a href="#">“Providing Target Information” on page 216</a>

## CWGetTargetStorage

Description	Retrieves global data for the active project target.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWGetTargetStorage( <a href="#">CWPluginContext</a> context, void** storage);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. storage         Returns a pointer to global storage retained for the plug-in with the current target by the IDE.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Use CWGetTargetStorage to retrieve target data previously stored using <a href="#">CWSetTargetStorage</a> . Plug-ins use <a href="#">CWSetTargetStorage</a> and CWGetTargetStorage to maintain global data with the active target. Target storage is maintained in memory for the duration of one invocation of the IDE, but is not stored on disk.

**NOTE** To associate data persistently with individual files in a project, use [CWGetPluginData](#) and [CWStorePluginData](#).

---

`CWGetTargetStorage` is useful when caching data for symbol unmangling requests. Since compilation and unmangling occur as separate requests, data must be stored globally to support unmangling.

A plug-in normally calls `CWGetTargetStorage` and [CWSetTargetStorage](#) when it receives a [reqTargetLoaded](#) or [reqTargetUnloaded](#) request. Target storage should be allocated persistently by calling [CWAllocateMemory](#) with the `isPermanent` parameter set to true, and disposed by calling [CWFreeMemory](#).

The IDE only provides target storage to plug-ins which set the [kCompUsesTargetStorage](#) or [linkerUsesTargetStorage](#) flags in their dropin flags. Target storage services will fail if the appropriate flag is not set.

See Also

- [“CWSetTargetStorage” on page 266](#)
- [“reqTargetLoaded” on page 320](#)
- [“reqTargetUnloaded” on page 321](#)
- [“CWGetPluginData” on page 129](#)
- [“CWStorePluginData” on page 145](#)
- [“CWAllocateMemory” on page 106](#)
- [“CWFreeMemory” on page 113](#)
- [“Specifying Plug-in Capabilities” on page 65](#)
- [“Compiler and Linker DropIn Flags” on page 199](#)

## **CWGetWorkingDirectory**

---

Description	Returns the location specified by the user in the <b>Working Directory</b> field of the <b>Runtime Settings</b> panel.
Prototype	#include <DropInCompilerLinker.h>

---

```
CW_CALLBACK CWGetWorkingDirectory(
    CWPluginContext context,
    CWFfileSpec* workingDirectorySpec);
```

Parameters	Parameters for this function are:
	context    Private, opaque IDE state.
	workingDirectorySpec                          Returns the user specified location entered in the <b>Working Directory</b> field of the <b>Runtime Settings</b> panel.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Linker plug-ins should call <code>CWGetWorkingDirectory</code> to determine the preferred working directory to establish prior to launching a target executable. <code>CWGetWorkingDirectory</code> returns the location specified in the <b>Working Directory</b> field of the <b>Runtime Settings</b> panel.
Mac OS	This routine is supported on Mac OS hosted IDEs, but only returns useful information when the current linker enables the <b>Runtime Settings</b> panel. Currently, this happens for the Java linker.
See Also	<a href="#">“CWGetCommandLineArgs” on page 244</a> <a href="#">“CWGetEnvironmentVariable” on page 244</a> <a href="#">“CWGetEnvironmentVariableCount” on page 246</a>

## CWIsAutoPrecompiling

Description	Indicates whether the current precompile operation was initiated by a build operation.
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; CW_CALLBACK CWIsAutoPrecompiling(     <a href="#">CWPluginContext</a> context,     Boolean*                                        isAutoPrecompiling);</pre>
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
isAutoPrecompiling	Returns true if the file currently being compiled should be precompiled, and the compile request resulted from a build operation.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	When responding to a <code>reqCompile</code> request, a compiler plug-in should call <code>CWIIsAutoPrecompiling</code> . If <code>CWIIsAutoPrecompiling</code> returns true, the plug-in should precompile the current source text obtained by calling <a href="#">CWGetMainFileText</a> , or by loading the text of the file indicated by <a href="#">CWGetMainFileName</a> . The resulting precompiled header should be stored in the location specified by <a href="#">CWGetPrecompiledHeaderSpec</a> .
	<code>CWIIsAutoPrecompiling</code> returns true when the user chooses <b>Project &gt; Make or Bring Up To Date</b> and a precompiled interface file must be updated. The IDE automatically precompiles only those file types for which the ‘precompiled’ flag is set in <b>Project Settings &gt; File Mappings</b> .
<b>NOTE</b>	<code>CWIIsAutoPrecompiling</code> returns true <i>only</i> when precompiling interface files encountered during an automated build operation, rather than when manually precompiling a file.
See Also	<a href="#">“reqCompile” on page 313</a> <a href="#">“CWGetPrecompiledHeaderSpec” on page 251</a> <a href="#">“CWIIsPrecompiling” on page 261</a> <a href="#">“CWGetMainFileID” on page 247</a> <a href="#">“CWGetMainFileName” on page 248</a> <a href="#">“CWGetMainFileSpec” on page 249</a> <a href="#">“CWGetMainFileText” on page 249</a>

[“Compiling” on page 211](#)

## CWIsCachingPrecompiledHeaders

Description	Indicates whether the IDE supports precompiled header caching.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWIsCachingPrecompiledHeaders( <a href="#">CWPluginContext</a> context, Boolean* isCaching);
Parameters	Parameters for this function are:  context                 Private, opaque IDE state. isCaching                 Returns true if the IDE currently honors requests to cache precompiled headers, made by calling <a href="#">CWCachePrecompiledHeader</a> .
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Before calling <a href="#">CWCachePrecompiledHeader</a> , a plug-in should call CWIsCachingPrecompiledHeaders to make sure the IDE is able to cache precompiled header data.
See Also	<a href="#">“CWCachePrecompiledHeader” on page 240</a>

## CWIsGeneratingDebugInfo

Description	Indicates whether a plug-in should generate debugging information.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWIsGeneratingDebugInfo( <a href="#">CWPluginContext</a> context, Boolean* isGenerating);
Parameters	Parameters for this function are:  context                 Private, opaque IDE state. isGenerating                 Returns true if the plug-in should generate debugging information.

Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Compiler plug-ins responding to <a href="#">reqCompile</a> requests should call <code>CWIsGeneratingDebugInfo</code> to determine if the plug-in should generate debugging information. <code>CWIsGeneratingDebugInfo</code> returns true if the debugging flag (indicated by a bug icon next to a file’s name in the project window) is enabled for the file being compiled.
See Also	<a href="#">“reqCompile” on page 313</a> <a href="#">“reqLink” on page 315</a> <a href="#">“Generating Debugging Data” on page 222</a>

## CWIsPrecompiling

Description	Indicates whether a plug-in should precompile the current file.				
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; CW_CALLBACK CWIsPrecompiling(     <a href="#">CWPluginContext</a> context,     Boolean* isPrecompiling);</pre>				
Parameters	Parameters for this function are:				
	<table border="0" style="width: 100%;"> <tr> <td style="width: 30%;"><code>context</code></td><td>Private, opaque IDE state.</td></tr> <tr> <td><code>isPrecompiling</code></td><td>Returns true if the file currently being compiled should be precompiled.</td></tr> </table>	<code>context</code>	Private, opaque IDE state.	<code>isPrecompiling</code>	Returns true if the file currently being compiled should be precompiled.
<code>context</code>	Private, opaque IDE state.				
<code>isPrecompiling</code>	Returns true if the file currently being compiled should be precompiled.				
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>				
Remarks	<p>When responding to a <a href="#">reqCompile</a> request, a compiler plug-in should call <code>CWIsPrecompiling</code>. If <code>CWIsPrecompiling</code> returns true, the plug-in should precompile the source code specified by <a href="#">CWGetMainFileText</a> and store it in the file returned by <a href="#">CWGetPrecompiledHeaderSpec</a>.</p> <p><code>CWIsPrecompiling</code> returns true when the user chooses <b>Precompile</b> from the <b>Project</b> menu, or when the IDE automatically recompiles a precompiled header file. In the latter case only, <a href="#">CWIsAutoPrecompiling</a> will also return true.</p>				

- 
- See Also    [“CWIsAutoPrecompiling” on page 258](#)  
[“CWGetPrecompiledHeaderSpec” on page 251](#)  
[“CWGetMainFileText” on page 249](#)  
[“reqCompile” on page 313](#)  
[“Compiling” on page 211](#)

## CWIsPreprocessing

Description	Indicates whether a plug-in should preprocess the current file.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWIsPreprocessing( <a href="#">CWPluginContext</a> context, Boolean* isPreprocessing);
Parameters	Parameters for this function are:  context                          Private, opaque IDE state. isPreprocessing                 Returns true if the file currently being compiled should be preprocessed.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	When responding to a <a href="#">reqCompile</a> request, a compiler plug-in should call <a href="#">CWIsPreprocessing</a> to determine if the source code obtained by calling <a href="#">CWGetMainFileText</a> should be preprocessed.  Preprocessing usually involves textual substitution or source transformation. The result of preprocessing is normally displayed in a window using <a href="#">CWCreateNewTextDocument</a> .
See Also	<a href="#">“CWGetMainFileText” on page 249</a> <a href="#">“CWGetMainFileSpec” on page 249</a> <a href="#">“CWIsPrecompiling” on page 261</a> <a href="#">“CWCreateNewTextDocument” on page 108</a> <a href="#">“reqCompile” on page 313</a>

### “Compiling” on page 211

# CWLoadObjectData

Description	Retrieves object data stored with a file.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWLoadObjectData( <a href="#">CWPluginContext</a> context, long whichfile, <a href="#">CWMemHandle</a> * objectdata);
Parameters	Parameters for this function are:
	context                 Private, opaque IDE state.
	whichfile             Specifies the index of the file for which to load object data, in target link order.
	objectdata            Returns a <a href="#">CWMemHandle</a> containing the object data.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plugins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	To retrieve object data associated with a file in the active project target, use <code>CWLoadObjectData</code> . <code>CWLoadObjectData</code> will return the object data most recently stored for <code>whichfile</code> in a call to <a href="#"><code>CWStoreObjectData</code></a> .
	The data returned in <code>objectdata</code> may be object code, resource data, or some combination of both. The IDE has no knowledge of and makes no assumptions about the content of the <code>objectdata</code> handle. The format of the data is determined by agreement between compiler and linker.
	If a plug-in stores object data in a file external to the project, by specifying a value for the <code>objectfile</code> field of a <a href="#"><code>CWObjectData</code></a> structure when calling <a href="#"><code>CWStoreObjectData</code></a> , <code>CWLoadObjectData</code> will load the entire contents of the file into a <a href="#"><code>CWMemHandle</code></a> and return it to the plug-in.
	Before accessing the data returned by <code>CWLoadObjectData</code> , lock it using <a href="#"><code>CWLockMemHandle</code></a> . When finished using the data, dispose it by calling <a href="#"><code>CWFreeObjectData</code></a> .

To specify the first file in the active target, use 0. To specify the last file, use the count returned by [CWGetProjectFileCount](#) - 1.

See Also [“CWStoreObjectData” on page 267](#)

[“CWObjectData” on page 281](#)

[“CWLockMemHandle” on page 134](#)

[“CWFreeObjectData” on page 242](#)

[“CWGetProjectFileCount” on page 132](#)

## CWPutResourceFile

Description Prompts the user to select a file in which to store resources.

Prototype

```
#include <DropInCompilerLinker.h>
CW_CALLBACK CWPutResourceFile(
    CWPluginContext context,
    const char* prompt,
    const char* name,
    CWFileSpec* filespec);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

prompt A C string prompt appearing in a standard file save dialog box.

name A C string specifying the default name of the resource file.

filespec Returns a file specification for a user-selected resource file.

Return An error code listed in [“Result Codes for Compiler and Linker Plug-ins” on page 328](#) or [“Result Codes for Plug-ins” on page 193](#)

Remarks `CWPutResourceFile` displays a standard file dialog box and waits for the user to specify the name and location of a new resource file. If the user cancels, `CWPutResourceFile` returns [cwErrUserCancelled](#).

`CWPutResourceFile` is currently used by the Mac OS 68K and PowerPC linkers when compiling code resources for targets with the **Display Dialog** checkbox set, and the **Merge to File** checkbox cleared. Other resource compilers may find it useful as well.

**WARNING!**

Due to threading issues in all versions of CodeWarrior through Pro 5, linkers should not call this routine unless they support handling the [`reqTargetInfo`](#) request reentrantly, in which case they must set the [`linkerGetTargetInfoThreadSafe`](#) dropin flag.

---

See Also

[“CWGetResourceFile” on page 252](#)

## CWSetTargetInfo

Description	Sets properties of the current target.
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; CW_CALLBACK CWSetTargetInfo(     CWPluginContext context,     CWTarGetInfo* targetInfo);</pre>
Parameters	Parameters for this function are:  context              Private, opaque IDE state. targetInfo            Specifies properties of the current target.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	In response to a <a href="#"><code>reqTargetInfo</code></a> request, a linker plug-in should call <code>CWSetTargetInfo</code> . <code>CWSetTargetInfo</code> specifies many properties of the current link target, including target processor and OS, the final binary type, and target debugging information. plug-ins can obtain the current target settings using <a href="#"><code>CWGetTargetInfo</code></a> .
See Also	<a href="#">“CWSetTargetInfo” on page 265</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWTarGetInfo” on page 283</a> <a href="#">“reqTargetInfo” on page 319</a>

## CWSetTargetStorage

Description	Stores global data with the current target.
Prototype	#include <DropInCompilerLinker.h> CW_CALLBACK CWSetTargetInfo( <a href="#">CWPluginContext</a> context, void* storage);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. storage         Specifies a pointer to global data to be retained by the IDE with the current target for the current plug-in.
Return	An error code listed in <a href="#">“Result Codes for Compiler and Linker Plug-ins” on page 328</a> or <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	Use CWSetTargetStorage to store global data with the current target. plug-ins use <a href="#">CWGetTargetStorage</a> and <a href="#">CWSetTargetStorage</a> to maintain global data with the active target. Target storage is maintained in memory for the duration of one invocation of the IDE, but is not stored on disk.
<b>NOTE</b>	To associate data persistently with individual files in a project, use <a href="#">CWGetPluginData</a> and <a href="#">CWStorePluginData</a> .

---

CWSetTargetStorage is useful when caching data for symbol unmangling requests. Since compilation and unmangling occur as separate requests, data must be stored globally to support unmangling.

A plug-in normally calls [CWGetTargetStorage](#) and [CWSetTargetStorage](#) when it receives a [reqTargetLoaded](#) or [reqTargetUnloaded](#) request. Target storage should be allocated persistently by calling [CWAllocateMemory](#) with its isPermanent parameter set to true, and disposed by calling [CWFreeMemory](#).

The IDE only provides target storage to plug-ins which set the [kCompUsesTargetStorage](#) or [linkerUsesTargetStorage](#) flags in their dropin flags. Target storage services will fail if the appropriate flag is not set.

See Also [“CWGetTargetStorage” on page 256](#)

[“reqTargetLoaded” on page 320](#)

[“reqTargetUnloaded” on page 321](#)

[“CWGetPluginData” on page 129](#)

[“CWStorePluginData” on page 145](#)

[“CWAlocateMemory” on page 106](#)

[“CWFreeMemory” on page 113](#)

[“Specifying Plug-in Capabilities” on page 65](#)

[“Compiler and Linker DropIn Flags” on page 199](#)

## **CWStoreObjectData**

Description Stores object, resource, and browser data with a file in the current target, and optionally specifies source dependencies.

Prototype 

```
#include <DropInCompilerLinker.h>
CW_CALLBACK CWStoreObjectData(
    CWPluginContext    context,
    long                  whichfile,
    CWOBJECTDATA\*    objectdata);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

whichfile Specifies the file for which the IDE should store object data and dependencies, by index in target link order.

objectdata Specifies numerous properties of the object data to be stored, including the object code, browser data, and optionally dependencies.

Return An error code listed in [“Result Codes for Compiler and Linker Plug-ins” on page 328](#) or [“Result Codes for Plug-ins” on page 193](#)

Remarks	CWStoreObjectData stores data passed in objectdata with the current project target and associates it with the file whichfile, specified by index.  Compilers normally call CWStoreObjectData when finished responding to a <a href="#">reqCompile</a> request, to save both object code and browser data.  CWStoreObjectData can also be used to establish dependencies among arbitrary files in a project. Normally, file dependencies are established using <a href="#">CWFindAndLoadFile</a> . In some cases, specifying dependencies using <a href="#">CWFindAndLoadFile</a> may be inconvenient. This is commonly the case when adapting command line tools. In such cases, a list of dependencies can be stored using CWStoreObjectData.
See Also	<a href="#">“CWObjectData” on page 281</a>

## User Routines for Compiler and Linker Plug-ins

In addition to the standard plug-in entry points, compiler and linker plug-ins provide two additional entry points for specifying associated file types, and for providing target information to the IDE. They may optionally export additional symbol unmangling and browser symbol entry points.

For more about compiler and linker-specific entry points, see [“Compiler and Linker-Specific Entry Points” on page 206](#). For information about other plug-in entry points, see [“Informational Entry Points” on page 65](#) and [“Responding to IDE Requests” on page 75](#).

The following entry points are specific to compilers and linkers:

- [CWPlugin\\_GetDefaultMappingList Entry Point](#)
- [CWPlugin\\_GetTargetList Entry Point](#)
- [Helper\\_GetCompilerBrSymbols Entry Point](#)
- [Helper\\_Unmangle Entry Point](#)

## **CWPlugin\_GetDefaultMappingList Entry Point**

Description	Provides an informational entry point that the IDE uses to determine the file types to associate with a plug-in.		
Prototype	<code>CWPLUGIN_ENTRY (CWPlugin_GetDefaultMappingList) (const CWExtMapList** defaultMappingList)</code>		
Parameters	Parameters for this entry point are:  <table><tr><td><code>defaultMappingList</code></td><td>Returns a <code>CWExtMapList</code> to the IDE containing an array of <code>CWExtensionMapping</code> structures specifying file type, extension, and flags.</td></tr></table>	<code>defaultMappingList</code>	Returns a <code>CWExtMapList</code> to the IDE containing an array of <code>CWExtensionMapping</code> structures specifying file type, extension, and flags.
<code>defaultMappingList</code>	Returns a <code>CWExtMapList</code> to the IDE containing an array of <code>CWExtensionMapping</code> structures specifying file type, extension, and flags.		
Return	Usually, the plug-in should return the <code>cwNoErr</code> result code. See <a href="#">“Result Codes for Plug-ins” on page 193</a> .		
Remarks	This entry point is used by the IDE to obtain the default list of file mappings to associate with a plug-in. This entry point is required for compiler and linker plug-ins. Usually, this is implemented simply by returning a pointer to a constant <code>CWExtMapList</code> structure.  File mappings appear in the <b>File Mappings</b> panel of the <b>Project Settings</b> dialog. File mappings determine which types of files will be presented to a plug-in by the IDE for compiling or linking, and control which files may be added to a project.  File mappings are added to a project by the IDE when creating a new project completely from scratch (not from stationery), and when the user reverts to factory defaults.		
Mac OS	The file type codes returned in <code>defaultMappingList</code> are primarily useful on Mac OS.  On Mac OS, this routine is optional. In its place, a plug-in may provide an <a href="#">‘EMap’ Resource</a> , specifying the same information.		
See Also	<a href="#">“Specifying File Mappings” on page 206</a>		

## **CWPlugin\_GetTargetList Entry Point**

Description	Provides an informational entry point that the IDE uses to determine the target CPUs and operating systems supported by a plug-in.		
Prototype	<code>CWPLUGIN_ENTRY (CWPlugin_GetTargetList) (const CWTARGETLIST** targetList)</code>		
Parameters	Parameters for this entry point are:  <table><tr><td><code>targetList</code></td><td>Returns a <code>CWTARGETLIST</code> to the IDE containing two arrays of 4-character codes listing the CPUs and operating systems supported by a plug-in.</td></tr></table>	<code>targetList</code>	Returns a <code>CWTARGETLIST</code> to the IDE containing two arrays of 4-character codes listing the CPUs and operating systems supported by a plug-in.
<code>targetList</code>	Returns a <code>CWTARGETLIST</code> to the IDE containing two arrays of 4-character codes listing the CPUs and operating systems supported by a plug-in.		
Return	Usually, the plug-in should return the <code>cwNoErr</code> result code. See <a href="#">“Result Codes for Plug-ins” on page 193</a> .		
Remarks	This entry point is used by the IDE to obtain a list of the operating systems and CPUs supported by a plug-in. This entry point is required for compiler and linker plug-ins. Usually, this is implemented simply by returning a pointer to a constant <code>CWTARGETLIST</code> structure.  This list is used to determine which plug-ins are enabled. When a linker is chosen from the Linker popup in the <b>Target Settings</b> panel of <b>Project Settings</b> , the linker's supported CPU and OS are used to find other matching compiler plug-ins, which are then enabled for use in the <b>File Mappings</b> panel. Also, the preference panels for all enabled plug-ins are added to the <b>Project Settings</b> dialog.		
Mac OS	On Mac OS, this routine is optional. In its place, a plug-in may provide a <a href="#">‘Targ’ Resource</a> , specifying the same information.		
See Also	<a href="#">“Specifying Target Platforms” on page 208</a>		

## **Helper\_GetCompilerBrSymbols Entry Point**

Description	Provides an informational entry point that the IDE uses to obtain names of custom browser symbol types generated by a compiler plug-in.
Prototype	<pre>#include &lt;DropinCompilerLinker.h&gt; typedef CWPLUGIN_ENTRY     (*Helper_GetCompilerBrSymbols)</pre>

( [CWCompilerBrSymbolInfo](#)\* BrowseSymbolInfo );

Parameters	Parameters for this entry point are:
	BrowseSymbolInfo      Returns a list of browser symbol type names to the IDE.
Return	An error value described in <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	<p>The IDE calls a compiler's browser symbol entry point to obtain the human-readable, localized names of custom browser symbol types from the plug-in. The IDE passes this entry point a structure containing the compiler's target storage (if any has been allocated), and a field in which the plug-in returns a pointer to a list. The list associates internal and localized type names with the custom symbol type codes generated by the compiler.</p> <p>The IDE only calls this entry point if a compiler sets the kCompEmitsOwnBrSymbols dropin flag. The IDE calls a compiler's browser symbol entry point if the file mappings for a project enable the compiler, regardless of whether any files associated with the compiler appear in the active project. This entry point is only supported for compiler plug-ins.</p> <p>Compilers that generate custom browser symbols assign numeric values to symbols to indicate symbol type. The values must be assigned from the range reserved for plug-ins, starting with browseCompSymbolStart and increasing sequentially. The strings returned by this entry point provide type names for the custom symbol types (not individual symbols; symbol names appear in the browse data).</p> <p>For each custom symbol type, the <a href="#">CWCompilerBrSymbolInfo</a> structure returned by this entry point provides two pieces of information:</p> <ol style="list-style-type: none"><li>1. <b>symName</b>: The symbol type's internal name, used to group symbols having functionally equivalent type. This name should not be localized.</li><li>2. <b>symUIName</b>: The symbol type's localized, user-visible name, displayed in browser windows, used when selecting the type of symbol to view (function, global variable, class, etc.).</li></ol> <p>This information is returned in an array of <a href="#">CWCompilerBrSymbol</a> structures, with one entry for each custom type used by the plug-in</p>

when generating browser data. The first array entry corresponds to the first symbol type (assigned code `browseCompSymbolStart`), the second entry corresponds to the second symbol type (assigned code `browseCompSymbolStart + 1`), and so on.

---

<b>NOTE</b>	Although not evident from the declarations, the IDE treats the <a href="#">CWCompilerBrSymbolList</a> structure returned by reference as constant data.
-------------	---

---

If multiple compilers generate symbols with the same type name (`symName`), the symbols will be grouped together under the same heading in the IDE's browser catalog window.

Mac OS      68K-based non-CFM plug-ins implement this entry point by providing a code resource of type 'Dhlp' which must be named "Helper\_GetCompilerBrSymbols". The `main()` entry point for the code resource must be declared as above.

See also

- [“CWCompilerBrSymbolInfo” on page 276](#)
- [“CWCompilerBrSymbolList” on page 277](#)
- [“CWCompilerBrSymbol” on page 276](#)
- [“EBrowserItem” on page 589](#)
- [“Compiler Browser Symbol Entry Point” on page 209](#)
- [“Compiler and Linker DropIn Flags” on page 199](#)
- [“‘Dhlp’ Resource” on page 642](#)
- [“‘Flag’ Resource” on page 645.](#)

## **Helper\_Unmangle Entry Point**

Description	Provides an informational entry point that the IDE uses to unmangle browser symbols generated by compiler and linker plug-ins.
Prototype	<pre>#include &lt;DropinCompilerLinker.h&gt; typedef CWPLUGIN_ENTRY     (*Helper_Unmangle) (CWUnmangleInfo*);</pre>

Parameters	Parameters for this entry point are:
CWUnmangleInfo	Provides information about the symbol to be unmangled, and buffers for returning unmangled symbol names to the IDE.
Return	An error value described in <a href="#">“Result Codes for Plug-ins” on page 193</a>
Remarks	The IDE calls a linker’s symbol unmangling entry point to convert a mangled identifier name emitted in browser data into its human-readable form. The IDE only calls this entry point if the plug-in sets the <code>linkerUnmangles</code> flag in its dropin flags. This entry point is only supported for linker plug-ins.

The prototype for this entry point is

```
CWPLUGIN_ENTRY (Helper_Unmangle)  
    (CWUnmangleInfo*)
```

The [CWUnmangleInfo](#) structure passed to the plug-in specifies the following:

1. `targetStorage`: contains the plug-in’s target storage, often used to maintain additional information useful when unmangling symbols. This should be allocated in response to a previous [reqTargetLoaded](#) request, and information relevant to unmangling should be saved in target storage during compilation and linking.
2. `mangledName`: Provides the mangled symbol name.
3. `unmangleBuff`: Provides storage in which to return an unmangled name,
4. `unmangleBuffSize`: Specifies the size of the unmangled name to return, including NULL terminator character. Do not exceed this size when unmangling.
5. `browserClassID`: Indicates the type of the symbol whose name is to be unmangled.
6. `browserLang`: The plug-in should return an [ELanguage](#) language code in this field specifying symbol source language.

The IDE calls this entry point to unmangle browser symbols when displaying browser windows.

Mac OS    68K-based non-CFM plug-ins implement this entry point by providing a code resource of type 'Dhlp' which must be named "Helper\_Unmangle". The main() entry point for the code resource must be declared as above.

See also    ["CWUnmangleInfo" on page 290](#)

["reqTargetLoaded" on page 320](#)

["ELanguage" on page 590](#)

["Dhlp' Resource" on page 642](#)

## Data Structures for Compiler and Linker Plug-ins

This section describes the plug-in API data structures available for compilers, linkers, pre-linkers, and post-linker.

The data structures for compilers and linkers are:

- [CWBrowseOptions](#)
- [CWCompilerBrSymbolInfo](#)
- [CWCompilerBrSymbolList](#)
- [CWDependencyInfo](#)
- [CWCompilerBrSymbol](#)
- [CWExtensionMapping](#)
- [CWExtMapList](#)
- [CWOBJECTData](#)
- [CWTARGETInfo](#)
- [CWTARGETList](#)
- [CWUnmangleInfo](#)

### CWBrowseOptions

Description    Specifies what browser information compilers should generate.

Definition    

```
#include <DropInCompilerLinker.h>
typedef struct CWBrowseOptions {
    Boolean                recordClasses;
    Boolean                recordEnums;
```

```
Boolean           recordMacros;
Boolean           recordTypedefs;
Boolean           recordConstants;
Boolean           recordTemplates;
Boolean           recordUndefinedFunctions;
long              reserved1;
long              reserved2;
} CWBrowseOptions;
```

Fields    The fields in `CWBrowseOptions` are:

recordClasses	If true, classes and structured types should be recorded. Examples of classes are C++ classes and Pascal classes. Examples of structured type classes are C structs and unions, and Pascal records.
recordEnums	If true, enumerated types should be recorded.
recordMacros	If true, macros should be recorded.
recordTypedefs	If true, user-defined types other than those recorded for <code>recordClasses</code> should be recorded.
recordConstants	If true, constant definitions should be recorded.
recordTemplates	If true, templates should be recorded.
recordUndefinedFunctions	If true, routines that are declared but not defined should be recorded.
reserved1	These fields are reserved by the IDE. Do not modify their contents.
reserved2	

Remarks    This structure specifies the source code elements for which a compiler should generate browser information. This structure is obtained by calling [CWGetBrowseOptions](#).

---

**NOTE**    The IDE assumes that compilers will always record browser information for global variables and functions, in addition to any item

types enabled by [CWGetBrowseOptions](#). Thus, there are no flags for functions and global variables; they are always enabled.

See Also [“CWGetBrowseOptions” on page 242](#)

## **CWCompilerBrSymbol**

Description	Describes a compiler-specific browser symbol type.				
Definition	<pre>#include &lt;DropInCompilerLinker.h&gt; typedef struct CWCompilerBrSymbol {     char             symName[32];     char             symUIName[32]; } CWCompilerBrSymbol;</pre>				
Fields	The fields in CWBrowserBrSymbol are:  <table><tr><td>symName</td><td>Specifies an internal name for the browser symbol type. This name should not be localized.</td></tr><tr><td>symUIName</td><td>Specifies the localized, human-readable name of the browser symbol type. The IDE presents this symbol to the user.</td></tr></table>	symName	Specifies an internal name for the browser symbol type. This name should not be localized.	symUIName	Specifies the localized, human-readable name of the browser symbol type. The IDE presents this symbol to the user.
symName	Specifies an internal name for the browser symbol type. This name should not be localized.				
symUIName	Specifies the localized, human-readable name of the browser symbol type. The IDE presents this symbol to the user.				
Remarks	This data structure describes a compiler-specific browser symbol type. It is used in a <a href="#">CWCompilerBrSymbolList</a> data structure, returned to the IDE by compilers, via the <a href="#">Helper_GetCompilerBrSymbols Entry Point</a> .  Symbols are grouped in the browser window by symName. If multiple compilers generate symbols with the same symName, the symbols will appear under the same heading in the IDE's browser catalog window.				
See Also	<a href="#">“CWCompilerBrSymbolList” on page 277</a> <a href="#">“CWCompilerBrSymbolInfo” on page 276</a> <a href="#">“Helper_GetCompilerBrSymbols Entry Point” on page 270</a>				

## **CWCompilerBrSymbolInfo**

Description	Used to return names of custom browser symbol types to the IDE.
-------------	---

Definition	<pre>#include &lt;DropInCompilerLinker.h&gt; typedef struct CWCompilerBrSymbolInfo {     void* targetStorage;     <a href="#">CWCompilerBrSymbolList</a>* symList; } CWCompilerBrSymbolInfo;</pre>				
Fields	<p>The fields in CWBrowserBrSymbolInfo are:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;">targetStorage</td><td>Points to the plug-in's target data, if it's available. This should have been set up during a previous <a href="#">reqTargetLoaded</a> request.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;">symList</td><td>Returns a pointer to a list of browser symbol type names.</td></tr> </table>	targetStorage	Points to the plug-in's target data, if it's available. This should have been set up during a previous <a href="#">reqTargetLoaded</a> request.	symList	Returns a pointer to a list of browser symbol type names.
targetStorage	Points to the plug-in's target data, if it's available. This should have been set up during a previous <a href="#">reqTargetLoaded</a> request.				
symList	Returns a pointer to a list of browser symbol type names.				
Remarks	<p>The IDE passes this data structure to a compiler's <a href="#">Helper_GetCompilerBrSymbols Entry Point</a>. The plug-in is expected to return a pointer to a constant list of custom symbol type names, in symList. The IDE passes the plug-in's target storage in targetStorage (but this is usually not required to return symbol type names).</p>				
See Also	<a href="#">“Helper_GetCompilerBrSymbols Entry Point” on page 270</a>				

## CWCompilerBrSymbolList

Description	Contains a list of compiler-specific browser symbols.				
Definition	<pre>#include &lt;DropInCompilerLinker.h&gt; typedef struct CWCompilerBrSymbolList {     short count;     <a href="#">CWCompilerBrSymbol</a> items[1]; } CWCompilerBrSymbolList;</pre>				
Fields	<p>The fields in CWBrowserBrSymbolList are:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;">count</td><td>Specifies the number of symbols in the list.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;">items</td><td>Points to an array of browser symbol type names.</td></tr> </table>	count	Specifies the number of symbols in the list.	items	Points to an array of browser symbol type names.
count	Specifies the number of symbols in the list.				
items	Points to an array of browser symbol type names.				
Remarks	<p>This data structure contains an array of compiler browser symbol type name structures. Each <a href="#">CWCompilerBrSymbol</a> element in the array specifies the internal and user-visible names for a single custom type used by a compiler when generating browser data.</p>				

See Also    [“CWCompilerBrSymbol” on page 276](#)

[“Helper GetCompilerBrSymbols Entry Point” on page 270](#)

## CWDependencyInfo

Description    Specifies a file depended upon by the file for which object code is being stored in a call to [CWStoreObjectData](#).

Definition

```
#include <DropInCompilerLinker.h>
typedef struct CWDependencyInfo {
    long             fileIndex;
    CWFileSpec      fileSpec;
    short            fileSpecAccessType;
    short            dependencyType;
} CWDependencyInfo;
```

Fields    The fields in CWDependencyInfo are:

`fileIndex`    Specifies a file depended upon by the file whose object data is being stored, by index in target link order. Use -1 to specify the depended-upon file using `fileSpec` instead.

`fileSpec`    When `fileIndex` is -1, this field specifies the depended-upon file by `CWFileSpec` instead of by index.

`fileSpecAccessType`    Specifies how the IDE should find the file after the user has chosen a command like **Project > Re-search for Files**. Valid values are: [cwAccessPathRelative](#), [cwAccessAbsolute](#), [cwAccessFileName](#), and [cwAccessFileRelative](#). Ignored unless `fileIndex` is -1.

`dependencyType`    A [CWDependencyType](#) value that specifies how the dependent file depends on the depended-upon file.

Remarks    A plug-in compiler or linker uses the CWDependencyInfo structure to store dependency information for a file when calling

[CWStoreObjectData](#). The [CWOBJECTDATA](#) structure's dependencies field points to an array of elements of CWDependencyInfo type, each of which specifies one file depended upon by the file whose object code is being stored.

To specify that the first file in the target is depended on by the object data store 0 in fileIndex. To specify the last file, store [CWGetProjectFileCount](#)(...) - 1. Use -1 to specify the depended-upon file using the fileSpec field rather than by index. This is most useful when your plug-in finds other depended-upon files by iterating through target files using [CWGetFileInfo](#) (for example, the Metrowerks Java compiler does this).

See Also

[“CWDependencyType” on page 158](#)

[“CWGetProjectFileCount” on page 132](#)

[“CWOBJECTDATA” on page 281](#)

[“CWStoreObjectData” on page 267](#)

[“CWGetFileInfo” on page 120](#)

## **CWExtensionMapping**

Description Provides flags indicating how the IDE should handle files having a specified type and extension.

Prototype

```
#include <DropInCompilerLinker.h>
typedef struct CWExtensionMapping {
    CWDATA\_TYPE          type;
    char                extension[32];
    CompilerMappingFlags flags;
} CWExtensionMapping;
```

Fields The fields in CWExtensionMapping are:

type	Specifies the type of the file. Relevant on Mac OS only.
------	--

---

extension	Specifies the file extension, if any, as a C string.
flags	Specifies bit flags indicating how the IDE should handle the file. Valid values are <code>kPrecompile</code> , <code>kLaunchable</code> , <code>kRsrcfile</code> , and <code>kIgnored</code> . Set all other bits to zero.
Remarks	Plug-ins use structures of type <code>CWExtensionMapping</code> to return file mappings to the IDE via the file mapping entry point. File mapping entry points return an array of <code>CWExtensionMapping</code> structures in a <a href="#">CWExtMapList</a> . Each array entry specifies one kind of project file, by extension and Mac OS file type (when relevant), and flags indicating how files of this type should be handled by the IDE.
See Also	<a href="#">“CWExtMapList” on page 280</a>

## **CWExtMapList**

Description	Returns a list of file mappings for a plug-in to the IDE.
Prototype	<pre>#include &lt;DropInCompilerLinker.h&gt; typedef struct CWExtMapList {     short                  version;     short                  nMappings;     CWExtensionMapping*   mappings; } CWExtMapList;</pre>
Fields	The fields in <code>CWExtMapList</code> are:
version	Specifies the version of the <code>CWExtMapList</code> structure. Use <code>kCurrentCWExtMapListVersion</code> for the latest version.
nMappings	Specifies the number of <code>CWExtensionMapping</code> structures appearing in <code>mappings</code> .
mappings	An array of <code>CWExtensionMapping</code> structures, each of which specifies a single file type, by file type and extension, and its associated flags.
Remarks	Compiler and linker plug-ins return a list of file mappings to the IDE via the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> . The IDE calls this entry point when adding file mappings to a newly

created project, or when the user reverts to factory defaults. The `CWExtMapList` structure specifies the number of file mappings returned, and returns a pointer to an array of `CWExtensionMapping` structures. Each `CWExtensionMapping` structure specifies a file type handled by the plug-in, and flags indicating how the IDE should handle files of that type.

See Also

[“`CWExtensionMapping`” on page 279](#)

[“Specifying File Mappings” on page 206](#)

## CWObjectData

Description Describes information about the object code generated by a compiler.

Definition

```
#include <DropInCompilerLinker.h>
typedef struct CWObjectData {
    CWMemHandle          objectdata;
    CWMemHandle          browsedata;
    long                  reserved1;
    long                  codesize;
    long                  udatasize;
    long                  idatasize;
    long                  compiledlines;
    Boolean               interfaceChanged;
    long                  reserved2;
    void*                 compilecontext;
    CWDependencyInfo*   dependencies;
    short                 dependencyCount;
    CWFileSpec*         objectfile;
} CWObjectData;
```

Fields The fields in `CWObjectData` are:

<code>objectdata</code>	Contains the object code, resource data, or other data that was generated from the file. May be <code>NULL</code> to indicate that the data is stored in a file external to the project.
-------------------------	--

<code>browsedata</code>	Contains the browser records that were generated for the file. May be <code>NULL</code> if browser data is not supported.
-------------------------	---

---

<code>reserved1</code>	Reserved for future use; plug-ins should store 0 in this field.
<code>codesize</code>	Specifies the size, in bytes, of the object data that was generated from the file.
<code>udatasize</code>	Specifies the size, in bytes, of uninitialized data in the object data.
<code>idatasize</code>	Specifies the size, in bytes, of initialized data in the object data.
<code>compiledlines</code>	Specifies the number of lines of source code processed while generating the object data.
<code>interfaceChanged</code>	Set this to true if the external interface of the file that was compiled has changed. Any other target file with a dependency of type <code>cwInterfaceDependency</code> on that file will be recompiled.
<code>compilecontext</code>	Reserved for use by the IDE. plug-ins should set this to 0.
<code>dependencies</code>	Points to an optional array of files that rely on the object data. Elements in this array are of type <a href="#"><code>CWDependencyInfo</code></a> . Set to NULL to specify no dependencies.
<code>dependencyCount</code>	Specifies the number of elements in the dependencies array.
<code>objectfile</code>	Specifies the location of external object code to associate with the file. Ignored unless <code>objectdata</code> is NULL. Set this to NULL to specify no external object code file.
<b>Remarks</b>	A plug-in compiler or linker uses the <code>CWObjectData</code> data structure to store object code, resource data, browser data, and other information generated from a source file. The plug-in passes this data structure to the IDE when it calls <a href="#"><code>CWStoreObjectData</code></a> .  <code>objectdata</code> stores the object code, resource data, and possibly debugging information generated from a source file or imported

---

from a library file. If `NULL`, then the data is assumed to reside in an external object file, specified by `objectfile`.

`browsedata` stores the browser information generated for a file, consisting of symbolic information, such as variables and their types, functions and their parameters, and classes and their methods and data members. `browsedata` is optional, and may be `NULL`.

`codesize`, `udatasize`, and `idatasize` specify the sizes of any code, data, and initialized data emitted in the `objectdata` handle. Since the IDE knows nothing about the format of the `objectdata` handle, it cannot infer these sizes. Any and all of these may be 0. `compiledlines` indicates the number of lines of source compiled by the plug-in. These fields are simply displayed to the user in the project window and are provided for informational purposes only.

`interfaceChanged` indicates to the IDE whether the public interface of the compiled file has changed. If so, the IDE will recompile any dependent file having a dependency of type `cwInterfaceDependency`.

`dependencies` and `dependencyCount` specify a list of files that the compiled file depends upon. Note that setting this flag may cause files to be recompiled even when the content of `objectdata` does not change. The `dependencies` list may be `NULL`.

`objectfile` specifies the location of external object code associated with a file in place of data in `objectdata`. `objectfile` is ignored unless `objectdata` is `NULL`.

See Also

[“CWStoreObjectData” on page 267](#)

[“CWLoadObjectData” on page 263](#)

[“CWDependencyInfo” on page 278](#)

[“CWFindAndLoadFile” on page 110](#)

[“CWDependencyType” on page 158](#)

## **CWTargetInfo**

Description

Specifies information about a project target.

Definition

```
#include <DropInCompilerLinker.h>
typedef struct CWTTargetInfo {
    short          outputType;
    CWFileSpec    outfile;
    CWFileSpec    symfile;
    CWFileSpec    runfile;
    short          linkType;
    Boolean        canRun;
    Boolean        canDebug;
    CWDataType    targetCPU;
    CWDataType    targetOS;
#if CWPLUGIN_HOST == CWPLUGIN_HOST_MACOS
    OSType         outfileCreator;
    OSType         outfileType;
    OSType         debuggerCreator;
    OSType         runHelperCreator;
#endif
#if CWPLUGIN_HOST == CWPLUGIN_HOST_WIN32
    Boolean        runHelperIsRegKey;
    Boolean        debugHelperIsRegKey;
    char           args[512];
    char           runHelperName[512];
    Boolean        runHelperRequiresURL;
    char           reserved2;
    char           debugHelperName[512];
#endif
    CWFileSpec    linkAgainstFile;
} CWTTargetInfo;
```

Fields    The fields in CWTtargetInfo are:

outputType	Specifies what the linker creates: <a href="#">linkOutputNone</a> , <a href="#">linkOutputFile</a> , or <a href="#">linkOutputDirectory</a> .
outfile	Specifies the target output file produced by the linker.
symfile	Specifies the file that contains symbolic debugging information produced by the linker.

runfile	Specifies the file that should be launched or passed to the run helper when the user chooses <b>Project &gt; Run</b> . This specification may be the same file as outfile. If this file is an executable file, such as an application, the IDE launches it directly. Otherwise, the IDE asks the application specified by runHelperName or runHelperCreator to open the file specified by runFile.
linkType	Specifies the target linker model. Valid values are: <a href="#">exelinkageFlat</a> , <a href="#">exelinkageSegmented</a> , and <a href="#">exelinkageOverlay</a> .
canRun	If true, the IDE activates the <b>Run</b> command.
canDebug	If true, the IDE activates the <b>Debug</b> command.
targetCPU	The processor that the linker generates software for. Permissible values for this field are listed in <a href="#">“Constants for Compiler and Linker Plug-ins” on page 292</a> .
targetOS	The operating system that the linker generates software for. Permissible values for this field are listed in <a href="#">“Constants for Compiler and Linker Plug-ins” on page 292</a> .
outfileCreator	If outputType is <a href="#">linkOutputFile</a> , this field contains the creator of the linker's output file.
outfileType	Defined for Mac OS-hosted plug-ins only. If outputType is <a href="#">linkOutputFile</a> , this field contains the file type of the linker's output file.
debuggerCreator	Defined for Mac OS-hosted plug-ins only.

Specifies the signature of the application to be used to debug the linker's output file.

Defined for Mac OS-hosted plug-ins only.

**runHelperCreator**

Specifies the signature of the application needed to run the linker's output file.

Defined for Mac OS-hosted plug-ins only.

**runHelperIsRegKey**

If true, `runHelperName` is a registry key. If false, the IDE treats `runHelperName` as a file name.

**debugHelperIsRegKey**

If true, `debugHelperName` is a registry key. If false, the IDE treats `debugHelperName` as a file name.

**args**

A C string constructed by the plug-in, containing command line arguments, to be passed to the executable prior to launch.

Defined for Windows-hosted plug-ins only.

**runHelperName**

If `runHelperIsRegKey` is false, this specifies the name of any application required to assist in running the executable. If `runHelperIsRegKey` is true, this is the name of a Windows registry key, that specifies the full path of the application to use.

Defined for Windows-hosted plug-ins only.

**runHelperRequiresURL**

	<p>Set this flag if the run helper expects the specification of the file to run to be passed using URL file syntax. For example: 'file://f:/test/foobartest.jar'</p>
	<p>Defined for Windows-hosted plug-ins only.</p>
reserved2	<p>Reserved; do not use. Defined on Windows only.</p>
debugHelperName	<p>If debugHelperIsRegKey is false, this specifies the name of any application required to assist in debugging the executable. If debugHelperIsRegKey is true, this is the name of a Windows registry key, that specifies the full path of the application to use.</p>
	<p>Defined for Windows-hosted plug-ins only.</p>
linkAgainstFile	<p>Specifies the file that a project which includes this one should link into its final binary. This file may be the same as <code>outfile</code>, but need not be.</p>
Remarks	<p>A linker plug-in uses the <code>CWTTargetInfo</code> data structure to store information about its target executable. This data structure is used by the <a href="#">CWGetTargetInfo</a> and <a href="#">CWSetTargetInfo</a> calls to the IDE.</p> <p><code>outputType</code> and <code>outfile</code> specify the type and location of the linker's output. <code>symfile</code> specifies the location of the corresponding symbolics file, if any, used by a debugger. <code>runfile</code> specifies a file to be passed to the run helper application. If <code>runfile</code> is the same as <code>outfile</code>, the IDE simply launches the target executable. If <code>runfile</code> is specified, but there is no run-helper, the IDE launches the target application, and passes it the <code>runfile</code>.</p> <p><code>linkType</code> describes the organization and addressing model of code in the final executable produced by the linker. The IDE supports normal ("flat"), overlayed (used most commonly by embedded systems), and segmented (Mac 68K code resource) linkage models.</p>

`canRun` specifies whether the IDE should enable the **Project > Run** command for this target. `canDebug` specifies whether the IDE should enable the **Project > Debug** command for this target.

`targetCPU` and `targetOS` specify the CPU and operating system that the target runs on. These fields are used to enable other plug-in types, which are determined by the currently selected linker.

Appropriate values for these fields are defined in `DropinCompilerLinker.h`.

Windows	When true, <code>runHelperIsRegKey</code> and <code>debugHelperIsRegKey</code> indicate that <code>runHelperName</code> and <code>debugHelperName</code> are full paths specifying the run and debug helper applications, respectively. When false, <code>runHelperName</code> and <code>debugHelperName</code> are the names of registry keys specifying the full paths of the run and debug helper applications. <code>runHelperRequiresURL</code> indicates that the run helper wants its primary file argument (which refers to <code>runfile</code> ) specified using URL syntax.
---------	---

As an example, to invoke a run helper named ‘javavm’ on a `runfile` named `test.java` in directory `c:\testapp` with an `args` strings of ‘-a -b -c’, with the `runfile` specified using URL syntax, the IDE would construct this command line:

---

```
javavm file:///c:/testapp/test.java -a -b -c
```

---

`args` specifies command line arguments to be passed to the final executable when launched, either directly, or by way of the run helper application. These arguments are specified by the plug-in and returned to the IDE. Linkers typically obtain the command line arguments specified by the user in **Runtime Settings** using [CWGetCommandLineArgs](#), add any necessary additional command line arguments, and return the complete string of arguments to the IDE.

`linkAgainstFile` specifies the file produced by this target which should be linked into any parent projects which include the current project as a subproject. `linkAgainstFile` is not always the same as `outfile`. For example, this happens when compiling DLLs: the linker produces both a ‘.dll’ file and a ‘.lib’ file. The former contains the code, but the latter is used when linking projects that use the DLL library.

Mac OS    `outFileType` and `outfileCreator` specify the file type (usually but not always ‘APPL’) and the signature, respectively, for Mac OS application targets.

`debuggerCreator` specifies the signature of the application to use to debug the target executable, when the user selects **Project > Debug**. `runHelperCreator`. Specifies the signature of the run helper application to use assist in running the target executable.

---

**NOTE**    The definition of this structure is platform dependent.

---

See Also    [“CWGetTargetInfo” on page 255](#)  
[“CWSetTargetInfo” on page 265](#)  
[“linkOutputNone” on page 307](#)  
[“linkOutputFile” on page 308](#)  
[“linkOutputDirectory” on page 308](#)  
[“Constants for Compiler and Linker Plug-ins” on page 292](#)

## CWTarGetList

Description    Returns a list of supported CPUs and operating systems to the IDE.

Prototype    

```
#include <DropInCompilerLinker.h>
typedef struct CWTarGetList {
    short          version;
    short          cpuCount;
    CWDatatype* cpus;
    short          osCount;
    CWDatatype* oss;
} CWTarGetList;
```

Fields    The fields in CWTarGetList are:

version	Specifies the version of this structure (different versions contain different information). To use the latest version of this structure, specify <code>kCurrentCWTTargetListVersion</code> for this field.
cpuCount	Specifies the number of CPU codes in the <code>cpus</code> array.
cpus	An array of CPU codes, each specifying one target CPU.
osCount	Specifies the number of OS codes in the <code>oss</code> array.
oss	An array of OS codes, each specifying one target operating system.
Remarks	Plug-ins use a structure of type <code>CWTTargetList</code> to return information about the CPU and operating systems they support to the IDE. The IDE uses this information to match compilers and linkers and their associated preference panels with the currently selected target linker.  Valid CPU and operating system codes are defined in <code>DropInCompilerLinker.h</code> .
<b>NOTE</b>	Currently, it is assumed that only one of <code>cpuCount</code> and <code>osCount</code> will be greater than one. That is, plug-ins may indicate support for multiple CPUs or multiple operating systems, but not both.

See Also [“CWPlugin\\_GetTargetList Entry Point” on page 270](#)

[“Specifying Target Platforms” on page 208](#)

[“CWDataType” on page 157](#)

## **CWUnmangleInfo**

Description	Contains information required to unmangle a single compiler or linker symbol.
Definition	<pre>#include &lt;DropInCompilerLinker.h&gt; typedef struct CWUnmangleInfo {     void*           targetStorage;</pre>

```

const char* mangledName;
char* unmangleBuff;
long unmangleBuffSize;
unsigned short browserClassID;
unsigned char browserLang;
unsigned char filler1;
} CWUnmangleInfo;

```

**Fields** The fields in `CWUnmangleInfo` are:

<code>targetStorage</code>	Specifies the plug-in's target data, if it's available. If required, this data should have been previously set up in response to a <a href="#">reqTargetLoaded</a> request.
<code>mangledName</code>	Specifies the text containing the name to unmangle, formatted as a NULL-terminated C string.
<code>unmangleBuff</code>	Provides a buffer in which the plug-in should store the unmangled name as a NULL-terminated C string.
<code>unmangleBuffSize</code>	Specifies the size of <code>unmangleBuff</code> . The plug-in should not place text larger than this size in <code>unmangleBuff</code> , including the terminating NULL byte.
<code>browserClassID</code>	Specifies the <a href="#">EBrowserItem</a> value of the item to be unmangled (the item's type).
<code>browserLang</code>	Returns the <a href="#">ELanguage</a> value for the source language of the symbol. The plug-in should specify the same language that was used when generating the browser data.
<code>filler1</code>	Reserved; do not use.

**Remarks** The IDE uses this data structure when it calls a linker's unmangling entry point to translate a compiler or linker's internal symbol into its human-readable version.

The linker should examine the mangled name provided in `mangledName` and any information about the mangling process

that the compiler or linker has retained in `targetStorage`, and store an unmangled version of the symbol name in `unmangleBuff`. The length of the unmangled symbol should not exceed the size specified in `unmangleBuffSize`, including the terminating NULL byte.

`browserClassID` specifies the type of the symbol being unmangled. `browserLang` specifies the source language of the symbol, as emitted by the compiler while generating browser symbols. Possible values are defined in `CompilerMapping.h`.

See Also [“Helper Unmangle Entry Point” on page 272](#)

[“reqTargetLoaded” on page 320](#)

[“EBrowserItem” on page 589](#)

[“ELanguage” on page 590](#)

## Constants for Compiler and Linker Plug-ins

This section describes the constant and predefined values in the compiler/linker plug-in API for the CodeWarrior IDE on Mac OS.

These constants are:

- [cantDisassemble](#)
- [cwAccessAbsolute](#)
- [cwAccessPathRelative](#)
- [cwAccessFileName](#)
- [cwAccessFileRelative](#)
- [DROPINCOMPILERLINKERAPIVERSION](#)
- [exelinkageFlat](#)
- [exelinkageOverlay1](#)
- [exelinkageSegmented](#)
- [isPostLinker](#)
- [isPreLinker](#)
- [kCandisassemble](#)

- [kCanimport](#)
- [kCanprecompile](#)
- [kCanpreprocess](#)
- [kCompAllowDupFileNames](#)
- [kCompAlwaysReload](#)
- [kCompEmitsOwnBrSymbols](#)
- [kCompMultiTargAware](#)
- [kCompReentrant](#)
- [kCompRequiresFileListBuildStartedMsg](#)
- [kCompRequiresProjectBuildStartedMsg](#)
- [kCompRequiresSubProjectBuildStartedMsg](#)
- [kCompRequiresTargetBuildStartedMsg](#)
- [kCompRequiresTargetCompileStartedMsg](#)
- [kCompSavesDbqPreprocess](#)
- [kCompUsesTargetStorage](#)
- [kGeneratescode](#)
- [kGeneratesrsrscs](#)
- [kIgnored](#)
- [kIsMPAware](#)
- [kIspascal](#)
- [kLaunchable](#)
- [kPersistent](#)
- [kPrecompile](#)
- [kRsrcfile](#)
- [linkAllowDupFileNames](#)
- [linkAlwaysReload](#)
- [linkMultiTargAware](#)
- [linkOutputDirectory](#)
- [linkOutputFile](#)
- [linkOutputNone](#)
- [linkRequiresFileListBuildStartedMsg](#)

## Compiler and Linker Plug-in Reference

### Constants for Compiler and Linker Plug-ins

---

- [linkRequiresProjectBuildStartedMsg](#)
- [linkRequiresSubProjectBuildStartedMsg](#)
- [linkRequiresTargetBuildStartedMsg](#)
- [linkRequiresTargetLinkStartedMsg](#)
- [linkerDisasmRequiresPreprocess](#)
- [linkerGetTargetInfoThreadSafe](#)
- [linkerInitializeOnMainThread](#)
- [linkerSuggestsNonRecursiveAccessPaths](#)
- [linkerUnmangles](#)
- [linkerUsesCaseInsensitiveSymbols](#)
- [linkerUsesFrameworks](#)
- [linkerUsesTargetStorage](#)
- [linkerWantsPreRunRequest](#)
- [magicCapLinker](#)
- [reqCheckSyntax](#)
- [reqCompile](#)
- [reqCompDisassemble](#)
- [reqDisassemble](#)
- [req fileListBuildEnded](#)
- [req fileListBuildStarted](#)
- [reqLink](#)
- [req PreprocessForDebugger](#)
- [req ProjectBuildEnded](#)
- [req ProjectBuildStarted](#)
- [req SubProjectBuildEnded](#)
- [req SubProjectBuildStarted](#)
- [req TargetBuildEnded](#)
- [req TargetBuildStarted](#)
- [req TargetInfo](#)
- [req TargetLinkEnded](#)
- [req TargetLinkStarted](#)

- [reqTargetLoaded](#)
- [reqTargetPrefsChanged](#)
- [reqTargetUnloaded](#)
- [targetCPU68K](#)
- [targetCPUAny](#)
- [targetCPUEmbeddedPowerPC](#)
- [targetCPUi80x86](#)
- [targetCPUMips](#)
- [targetCPUNECv800](#)
- [targetCPUPowerPC](#)
- [targetOSAny](#)
- [targetOSEmbeddedABI](#)
- [targetOSMacintosh](#)
- [targetOSMagicCap](#)
- [targetOSOS9](#)
- [targetOSWindows](#)

## **cantDisassemble**

Description	Specifies that a linker does <i>not</i> support disassembly.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant is used in a linker's <a href="#">DropInFlags</a> to indicate that it does not support disassembly in response to a <a href="#">reqDisassemble</a> request.
<b>NOTE</b>	Unlike most dropin flags, setting this flag indicates that the plug-in does <i>not</i> support the corresponding feature.
<b>CAUTION</b>	Do not confuse <code>cantDisassemble</code> with <a href="#">kCanDisassemble</a> , which applies to compilers and has a different numeric value.
See Also	<a href="#">“Linker Capability Flags” on page 203</a>

## **cwAccessAbsolute**

Description	Specifies that a depended-upon file should be stored using its absolute path.
Definition	#include <DropInCompilerLinker.h>
Remarks	Use cwAccessPathRelative in a <a href="#">CWDependencyInfo</a> structure passed to <a href="#">CWStoreObjectData</a> to specify that the IDE should store the dependency using a full path. This is the least robust method of storing access paths in projects. Absolute paths break or point to the wrong location when a project is moved or copied.
See Also	<a href="#">“CWObjectData” on page 281</a> <a href="#">“CWStoreObjectData” on page 267</a>

## **cwAccessPathRelative**

Description	Specifies that a depended-upon file should be stored using a file path relative to one of the active target's access paths.
Definition	#include <DropInCompilerLinker.h>
Remarks	Use cwAccessPathRelative in a <a href="#">CWDependencyInfo</a> structure passed to <a href="#">CWStoreObjectData</a> to specify that the IDE should store the dependency using a path relative to one of the project paths. Relative path specifications have the advantage that they usually work correctly even when a project is moved or copied.
See Also	<a href="#">“CWObjectData” on page 281</a> <a href="#">“CWStoreObjectData” on page 267</a>

## **cwAccessFileName**

Description	Specifies that a depended-upon file should be stored using just its file name, not its path.
Definition	#include <DropInCompilerLinker.h>
Remarks	Use cwAccessPathRelative in a <a href="#">CWDependencyInfo</a> structure passed to <a href="#">CWStoreObjectData</a> to specify that the IDE should store the dependency using just a file's name, rather than its path.

This method of locating a file is likely to fail if a project contains more than one file by the same name.

## **cwAccessFileRelative**

Description	Specifies that a depended-upon file should be searched for using a path relative to the dependent file's path.
Definition	#include <DropInCompilerLinker.h>
Remarks	Use <i>cwAccessPathRelative</i> in a <a href="#">CWDependencyInfo</a> structure passed to <a href="#">CWStoreObjectData</a> to specify that the IDE should store the dependency using a path relative to the dependent file.
See Also	<a href="#">“CWObjectData” on page 281</a> <a href="#">“CWStoreObjectData” on page 267</a>

## **DROPINCOMPILERLINKERAPIVERSION**

Description	Specifies the current version of the compiler and linker plug-in API at compile time.
Definition	#include <DropInCompilerLinker.h>
Remarks	This predefined symbol specifies the current version of the compiler plug-in API. Use this value in a <a href="#">DropInFlags</a> structure to indicate that your plug-in supports the latest plug-in API version that was available at the time it was compiled.

---

<b>NOTE</b>	Any plug-in which uses this define will automatically specify the latest API version when the plug-in API headers change and the plug-in is recompiled. Usually, this is desirable. However, plug-ins that depend upon older API versions should use a different fixed constant to specify the latest supported API version.
-------------	--

---

## **exelinkageFlat**

Description	Specifies that the plug-in generates an executable binary consisting of a single piece of binary code.
Definition	#include <DropInCompilerLinker.h>

Remarks	This predefined symbol, used in the linkType field of the <a href="#">CWTargetInfo</a> data structure, specifies that the linker generates executables composed of one contiguous piece of executable code, rather than multiple, independently loaded code segments or overlays.  For targets specifying this linkage type, the IDE displays files in the link order project view in a single list, without any subgroups.
See Also	<a href="#">“CWTargetInfo” on page 283</a>

## exelinkageOverlay1

Description	Specifies that the plug-in generates an executable binary comprised of overlays.
Definition	#include <DropInCompilerLinker.h>
Remarks	This predefined symbol, used in the linkType field of the <a href="#">CWTargetInfo</a> data structure, specifies that the linker organizes its executable file into overlays. Overlays are code images loaded into the same portion of physical memory under program control, to reduce memory requirements.  For targets that specify this linkage type, the IDE displays files in the link order project view as overlays. This view allows files to be grouped into overlays, and overlays to be grouped into overlay groups.
See Also	<a href="#">“CWTargetInfo” on page 283</a>

## exelinkageSegmented

Description	Specifies that the plug-in generates executable code consisting of separate code resources.
Definition	#include <DropInCompilerLinker.h>
Remarks	This predefined symbol, used in the linkType field of the <a href="#">CWTargetInfo</a> data structure, specifies that the linker organizes an executable file into code segments. This is currently only used on Mac OS.  For targets that specify this linkage type, the IDE displays files in the link order project view grouped into segments.

See Also [“CWTarGetInfo” on page 283](#)

## **isPostLinker**

Description Specifies that a plug-in is a post-linker.

Definition #include <DropInCompilerLinker.h>

Remarks This constant is used in a linker’s [DropInFlags](#) to indicate that it is a post-linker which receives IDE requests after the target linker has been run.

See Also [“Linker Capability Flags” on page 203](#)

## **isPreLinker**

Description Specifies that a linker plug-in is a pre-linker.

Definition #include <DropInCompilerLinker.h>

Remarks This constant is used in a linker’s [DropInFlags](#) to indicate that it is a pre-linker which receives IDE requests before the target linker runs.

See Also [“Linker Capability Flags” on page 203](#)

## **kCandisassemble**

Description Specifies that a compiler supports code disassembly.

Definition #include <CompilerMapping.h>

Remarks This constant is used in a compiler’s [DropInFlags](#) to indicate that it supports code disassembly, in response to a [reqCompDisassemble](#) request.

---

**CAUTION** Do not confuse `kCandisassemble` with `cantDisassemble`, which applies to compilers, and has a different numeric value.

---

See Also [“Compiler Capability Flags” on page 200](#)

## **kCanimport**

Description Specifies that a compiler can import a compiled object code library.

Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that it imports compiled object code by adding it to the project in response to a <a href="#">reqCompile</a> request. When this flag is set, the IDE enables the <b>Weak Link</b> checkbox in the project inspector for the file types associated with the compiler.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## kCanprecompile

Description	Specifies that a compiler can precompile one or more of the source file types it handles.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that the compiler supports precompilation of interface files, to accelerate the compilation process. To support precompilation, plug-ins must also mark the relevant file types as precompilable.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>
	<a href="#">"Specifying File Mappings" on page 206</a>

## kCanpreprocess

Description	Specifies that a compiler can preprocess one or more of the source file types it handles.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that the compiler performs textual substitution or expansion on one or more of its file types, and presents the results to users in a window in response to a <a href="#">reqPreprocess</a> request.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## kCompAllowDupFileNames

Description	Specifies that a compiler can properly handle the appearance of multiple files in one target having the same name.
-------------	--

Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that the compiler can handle project targets containing multiple files having the same name (but different paths). Most compilers should set this flag. In the past, some compilers used file names to construct unique symbol names, which caused problems when multiple files having the same name appeared in one target.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## **kCompAlwaysReload**

Description	Specifies that the IDE should reload the compiler before every request.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that the compiler needs to be reloaded between requests from the IDE (except reqInitialize and reqTerminate). This setting is useful for compilers that have complex global state that must be reinitialized to handle requests properly. This is typically used when porting existing (often command line) tools that make heavy use of global state.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## **kCompEmitsOwnBrSymbols**

Description	Specifies that a compiler emits browser symbols of specialized type.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that the browser symbol information generated by a compiler contains custom type codes, indicating the presence of nonstandard symbol types. When set, the IDE will call the compiler's browser symbol entry point, to obtain the names of the custom symbol types.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a> <a href="#">"Helper_GetCompilerBrSymbols Entry Point" on page 270</a>

## kCompMultiTargAware

Description	Unused.
Definition	#include <CompilerMapping.h>
Remarks	Do not use this flag. It is not currently functional.
See Also	<a href="#">“Compiler Capability Flags” on page 200</a>

## kCompReentrant

Description	Specifies that the compiler is reentrant and permits the IDE to issue commands to multiple instances simultaneously.
Definition	#include <CompilerMapping.h>
Remarks	This flag enables the Concurrent Compiles facility.
See Also	<a href="#">“Compiler Capability Flags” on page 200</a>

## kCompRequiresFileListBuildStartedMsg

Description	Specifies that a compiler requires the reqFileListBuildStarted and reqFileListBuildEnded requests.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler’s <a href="#">DropInFlags</a> to indicate that a compiler wants to receive reqFileListBuildStarted and reqFileListBuildEnded requests.
See Also	<a href="#">“Compiler Capability Flags” on page 200</a>

## kCompRequiresProjectBuildStartedMsg

Description	Specifies that a compiler requires the reqProjectBuildStarted and reqProjectBuildEnded requests.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler’s <a href="#">DropInFlags</a> to indicate that a compiler wants to receive reqProjectBuildStarted and reqProjectBuildEnded requests.

See Also    [“Compiler Capability Flags” on page 200](#)

## **kCompRequiresSubProjectBuildStartedMsg**

Description	Specifies that a compiler requires the <code>reqSubProjectBuildStarted</code> and <code>reqSubProjectBuildEnded</code> requests.
Definition	<code>#include &lt;CompilerMapping.h&gt;</code>
Remarks	This constant is used in a compiler’s <a href="#">DropInFlags</a> to indicate that a compiler wants to receive <code>reqSubProjectBuildStarted</code> and <code>reqSubProjectBuildEnded</code> requests.
See Also	<a href="#">“Compiler Capability Flags” on page 200</a>

## **kCompRequiresTargetBuildStartedMsg**

Description	Specifies that a compiler requires the <code>reqTargetBuildStarted</code> and <code>reqTargetBuildEnded</code> requests.
Definition	<code>#include &lt;CompilerMapping.h&gt;</code>
Remarks	This constant is used in a compiler’s <a href="#">DropInFlags</a> to indicate that a compiler wants to receive <code>reqTargetBuildStarted</code> and <code>reqTargetBuildEnded</code> requests.
See Also	<a href="#">“Compiler Capability Flags” on page 200</a>

## **kCompRequiresTargetCompileStartedMsg**

Description	Indicates that the compiler can receive target compile started and ended messages.
Definition	<code>#include &lt;CompilerMapping.h&gt;</code>
Remarks	A compiler that recognizes started and ended messages can benefit from pre-work or post-processing after all individual compilations have been completed.
See Also	<a href="#">“Compiler Capability Flags” on page 200</a>

## **kCompSavesDbgPreprocess**

Description	Obsolete. Do not use.
-------------	-----------------------

## kCompUsesTargetStorage

Description	Specifies that a compiler uses target storage services.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that a compiler uses target storage facilities ( <a href="#">CWGetTargetStorage</a> and <a href="#">CWSetTargetStorage</a> ). Target storage is used to store global data that a plug-in wishes to preserve even when the plug-in is unloaded by the IDE.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## kGeneratescode

Description	Specifies that a compiler generates code.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that a compiler generates object code in response to a <a href="#">reqCompile</a> request, and stores it in the project using <a href="#">CWStoreObjectData</a> . The only compilers that should not set this flag are those that generate source code or resources exclusively.
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## kGeneratesrsrcs

Description	Specifies that a compiler generates resource data.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a compiler's <a href="#">DropInFlags</a> to indicate that a compiler generates binary resource data in response to a <a href="#">reqCompile</a> request, and stores it in the project using <a href="#">CWStoreObjectData</a> .
See Also	<a href="#">"Compiler Capability Flags" on page 200</a>

## kIgnored

Description	Specifies that a file type should be ignored by the IDE during <b>Make</b> operations.
-------------	--

Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a plug-in's file mappings to indicate that the corresponding file type should be completely ignored by the IDE during build operations.
See Also	<a href="#">“CWPlugin_GetDefaultMappingList Entry Point” on page 269</a>

## **kLaunchable**

Description	Specifies a file type that may be launched directly from the project window.
Definition	#include <CompilerMapping.h>
Remarks	This constant is used in a plug-in's file mappings to indicate that the IDE should launch or open files of this type when double-clicked by the user in the project window.

See Also [“CWPlugin\\_GetDefaultMappingList Entry Point” on page 269](#)

## **kIsMPAware**

Description	Currently unused.
Definition	#include <CompilerMapping.h>
Remarks	This flag is not currently functional; do not use.

See Also [“Compiler Capability Flags” on page 200](#)

## **kIsPascal**

Description	Set by the Metrowerks Pascal compiler.
Definition	#include <CompilerMapping.h>
Remarks	This constant should not be used by third party plug-ins.

See Also [“Compiler Capability Flags” on page 200](#)

## **kPersistent**

Description	Specifies that the IDE should avoid unloading the compiler.
Definition	#include <CompilerMapping.h>

**Remarks** This constant is used in a compiler's [DropInFlags](#) to indicate that the compiler wishes to be loaded and unloaded as infrequently as possible. Setting this flag does *not* guarantee that a plug-in will remain resident for the duration of an IDE run. This is most commonly used for plug-ins that perform complex initialization tasks, to reduce the time spent initializing plug-in state.

**See Also** [“Compiler Capability Flags” on page 200](#)

## **kPrecompile**

**Description** Specifies that a file type may be precompiled.

**Definition** #include <CompilerMapping.h>

**Remarks** This constant is used in a plug-in's file mappings to indicate that the corresponding file type can be precompiled by the plug-in. plug-ins must set this flag to receive `reqPrecompile` requests

**See Also** [“CWPlugin\\_GetDefaultMappingList Entry Point” on page 269](#)

## **kRsrcfile**

**Description** Specifies that a file type is a binary resource file.

**Definition** #include <CompilerMapping.h>

**Remarks** This constant is used in a plug-in's file mappings to indicate that the corresponding file type is a binary resource file, to be included in final compiled executables. Typically only used on Mac OS.

**See Also** [“CWPlugin\\_GetDefaultMappingList Entry Point” on page 269](#)

## **linkAllowDupFileNames**

**Description** Specifies that a linker properly handles targets containing multiple files having the same name.

**Definition** #include <DropInCompilerLinker.h>

**Remarks** This constant is used in a linker's [DropInFlags](#) to indicate that a linker supports projects containing multiple files with the same name. Most linkers should set this flag. An exception would be linkers which use a file's name to help construct unique symbol names.

See Also [“Linker Capability Flags” on page 203](#)

[Specifying File Mappings](#)

## linkAlwaysReload

Description Specifies that the IDE should reload the linker before every request.

Definition 

```
#include <DropInCompilerLinker.h>
```

Remarks This constant is used in a linker’s [DropInFlags](#) to indicate that the linker needs to be reloaded between requests from the IDE (except reqInitialize and reqTerminate). This setting is useful for linkers that have complex global state that must be reinitialized to handle requests properly. This is typically used when porting existing (often command line) tools that make heavy use of global state.

See Also [“Linker Capability Flags” on page 203](#)

## linkMultiTargAware

Description Specifies that a linker wishes to be reloaded every time the project target changes.

Definition 

```
#include <DropInCompilerLinker.h>
```

Remarks This constant is used in a linker’s [DropInFlags](#) to indicate that a linker wants to receive reqInitialize and reqTerminate requests every time the current project target changes. When this flag is *clear*, the IDE sends the requests; when set, it does *not* send the requests.

See Also [“Linker Capability Flags” on page 203](#)

## linkOutputNone

Description Specifies that the linker produces no final target binaries.

Definition 

```
#include <DropInCompilerLinker.h>
```

Remarks This predefined symbol, used in the outputType field of the [CWTtargetInfo](#) data structure, specifies that the linker produces no binary output file.

See Also [“CWTargetInfo” on page 283](#)

## linkOutputFile

Description Specifies that the linker produces a file.

Definition `#include <DropInCompilerLinker.h>`

Remarks This predefined symbol, used in the `outputType` field of the [CWTargetInfo](#) data structure, specifies that the linker produces a file.

See Also [“CWTargetInfo” on page 283](#)

## linkOutputDirectory

Description Specifies that the linker produces a directory and one or more files within the directory.

Definition `#include <DropInCompilerLinker.h>`

Remarks This predefined symbol, used in the `outputType` field of the [CWTargetInfo](#) data structure, specifies that the linker produces a directory containing multiple target files.

See Also [“CWTargetInfo” on page 283](#)

## linkRequiresFileListBuildStartedMsg

Description Specifies that a linker requires `reqFileListBuildStarted` and `reqFileListBuildEnded` requests.

Definition `#include <DropInCompilerLinker.h>`

Remarks This constant is used in a linker’s [DropInFlags](#) to indicate that a linker wants to receive `reqFileListBuildStarted` and `reqFileListBuildEnded` requests.

See Also [“Linker Capability Flags” on page 203](#)

[“Additional Compiler and Linker Requests” on page 223](#)

## **linkRequiresProjectBuildStartedMsg**

Description	Specifies that a linker requires <code>reqProjectBuildStarted</code> and <code>reqProjectBuildEnded</code> requests.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant is used in a linker's <a href="#">DropInFlags</a> to indicate that a linker wants to receive <code>reqProjectBuildStarted</code> and <code>reqProjectBuildEnded</code> requests.
See Also	<a href="#">“Linker Capability Flags” on page 203</a> <a href="#">“Additional Compiler and Linker Requests” on page 223</a>

## **linkRequiresSubProjectBuildStartedMsg**

Description	Specifies that a linker requires <code>reqSubProjectBuildStarted</code> and <code>reqSubProjectBuildEnded</code> request.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant is used in a linker's <a href="#">DropInFlags</a> to indicate that a linker wants to receive <code>reqSubProjectBuildStarted</code> and <code>reqSubProjectBuildEnded</code> requests.

[“Linker Capability Flags” on page 203](#)

[“Additional Compiler and Linker Requests” on page 223](#)

## **linkRequiresTargetBuildStartedMsg**

Description	Specifies that a linker requires <code>reqTargetBuildStarted</code> and <code>reqTargetBuildEnded</code> request.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant is used in a linker's <a href="#">DropInFlags</a> to indicate that a linker wants to receive <code>reqTargetBuildStarted</code> and <code>reqTargetBuildEnded</code> requests.

[“Linker Capability Flags” on page 203](#)

[“Additional Compiler and Linker Requests” on page 223](#)

## linkRequiresTargetLinkStartedMsg

Description Specifies that a linker requires `reqTargetLinkStarted` and `reqTargetLinkEnded` request.

Definition `#include <DropInCompilerLinker.h>`

Remarks This constant is used in a linker's [DropInFlags](#) to indicate that a linker wants to receive `reqTargetLinkStarted` and `reqTargetLinkEnded` requests.

See Also [“Linker Capability Flags” on page 203](#)

[“Additional Compiler and Linker Requests” on page 223](#)

## linkerDisasmRequiresPreprocess

Description Obsolete. Do not use.

## linkerGetTargetInfoThreadSafe

Description Specifies that a linker implements the [reqTargetInfo](#) request in a thread-safe manner.

Definition `#include <DropInCompilerLinker.h>`

Remarks This constant is used in a linker's [DropInFlags](#) to indicate that the linker is thread safe or not.

See Also [“Linker Capability Flags” on page 203](#)

[“Providing Target Information” on page 216](#)

## linkerInitializeOnMainThread

Description Specifies that a linker must initialize on the main thread.

Definition `#include <DropInCompilerLinker.h>`

Remarks This constant is used in a linker's [DropInFlags](#) to indicate that the linker must initialize on the main thread.

See Also [“Linker Capability Flags” on page 203](#)

## **linkerSuggestsNonRecursiveAccessPaths**

Description	When off, the IDE creates a new access path. When on, it recurses access paths.
Definition	#include <CompilerMapping.h>
Remarks	On Mac and Windows, the default is for new paths to have the recursive flag on.
See Also	<a href="#">“Linker Capability Flags” on page 203</a>

## **linkerUnmangles**

Description	Specifies that a linker supports name unmangling.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant is used in a linker’s <a href="#">DropInFlags</a> to indicate that a linker exports a name unmangling entry point, for converting mangled identifiers back into human-readable form.
See Also	<a href="#">“Linker Capability Flags” on page 203</a>
	<a href="#">“Linker Symbol Unmangling Entry Point” on page 210</a>

## **linkerUsesCaseInsensitiveSymbols**

Description	If this flag is on, browser symbols are not sensitive to case. If it is off, browser symbols are case-sensitive.
Definition	#include <CompilerMapping.h>
Remarks	Use this flag to specify whether to use case-sensitive or -insensitive browser symbols.
See Also	<a href="#">“Linker Capability Flags” on page 203</a>

## **linkerUsesFrameworks**

Description	Setting this flag enables IDE handling of frameworks in the MacOS X Mach-o target.
Definition	#include <CompilerMapping.h>
Mac OS	This flag only applies to the mach-o target.

Remarks      Use this flag to specify whether the IDE should handle frameworks in the MacOS X Mach-o target.

See Also     [“Linker Capability Flags” on page 203](#)

## **linkerUsesTargetStorage**

Description     Specifies that a linker uses target storage.

Definition    `#include <DropInCompilerLinker.h>`

Remarks      This constant is used in a linker’s [DropInFlags](#) to indicate that the linker uses target storage, by calling [CWGetTargetStorage](#) and [CWSetTargetStorage](#).

See Also     [“Linker Capability Flags” on page 203](#)

[“CWGetTargetStorage” on page 256](#)

[“CWSetTargetStorage” on page 266](#)

[“Managing Target Storage” on page 230](#)

## **linkerWantsPreRunRequest**

Description     Specifies that a linker wants to receive a reqPreRun request.

Definition    `#include <DropInCompilerLinker.h>`

Remarks      This constant is used in a linker’s [DropInFlags](#) to indicate that it want to receive the [reqPreRun](#) request, prior to launch of the target executable.

See Also     [“Linker Capability Flags” on page 203](#)

## **magicCapLinker**

Description     Obsolete. do not use.

## **reqCheckSyntax**

Description     Reserved for use by Metrowerks.

Definition    `#include <DropInCompilerLinker.h>`

---

Remarks	This request is currently unused. The IDE does <i>not</i> issue this request when the user selects <b>Project &gt; Check Syntax</b> . Instead, plug-ins receive a <a href="#">reqCompile</a> request.
See Also	<a href="#">“Compiling” on page 211</a> <a href="#">“reqCompile” on page 313</a>

## reqCompile

Description	Asks a compiler plug-in to compile a project file into object data, resource data, or source code text.
Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request when compiling a source file, as part of a <b>Project &gt; Compile, Make, Bring Up To Date, Check Syntax, Preprocess, or Precompile</b> operation. To determine which file should be compiled, use <a href="#">CWGetMainFileID</a> , <a href="#">CWGetMainFileSpec</a> , or <a href="#">CWGetMainFileText</a> .
See Also	<a href="#">“Compiling” on page 211</a> <a href="#">“CWGetMainFileID” on page 247</a> <a href="#">“CWGetMainFileSpec” on page 249</a> <a href="#">“CWGetMainFileText” on page 249</a> <a href="#">“CWStoreObjectData” on page 267</a> <a href="#">“CWGetPluginRequest” on page 130</a> <a href="#">“CWDonePluginRequest” on page 109</a>

## reqCompDisassemble

Description	Asks a compiler plug-in to convert binary object code into human-readable text.
Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request when the user chooses <b>Project &gt; Disassemble</b> . To determine which file should be disassembled, use <a href="#">CWGetMainFileID</a> or <a href="#">CWGetMainFileSpec</a> .

The IDE only sends this request if the compiler plug-in specifies that it handles disassembly in its 'Flag' resource by setting the [kCanDisassemble](#) flag.

See Also

- [“Disassembling” on page 219](#)
- [“CWGetMainFileID” on page 247](#)
- [“CWGetMainFileSpec” on page 249](#)
- [“'Flag' Resource” on page 645](#)
- [“CWGetPluginRequest” on page 130](#)
- [“CWDonePluginRequest” on page 109](#)

## **reqDisassemble**

Description      Asks a linker to generate human-readable text from object data.

Definition      `#include <DropInCompilerLinker.h>`

The IDE sends this request when the user chooses **Project > Disassemble**. To determine which file should be disassembled, use [CWGetMainFileID](#) or [CWGetMainFileSpec](#).

The IDE only sends this request if the linker plug-in specifies that it handles disassembly in its 'Flag' resource by setting the [cantDisassemble](#) flag.

To determine which file that must be disassembled, use [CWGetMainFileID](#). To obtain the object code for the file to be disassembled, use [CWLoadObjectData](#).

See Also

- [“Disassembling” on page 219](#)
- [“CWGetMainFileID” on page 247](#)
- [“CWLoadObjectData” on page 263](#)
- [“CWGetPluginRequest” on page 130](#)
- [“CWDonePluginRequest” on page 109](#)

## reqFileListBuildEnded

Description	The IDE sends this request at the end of a <b>Compile</b> operation.
Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request when it has finished compiling one or more files manually compiled by the user (that is, files compiled using <b>Compile</b> , rather than <b>Make</b> ).  The IDE sends this request only once, at the end of a <b>Compile</b> operation.
See Also	<a href="#">“Additional Compiler and Linker Requests” on page 223</a> <a href="#">“reqFileListBuildStarted” on page 315</a> <a href="#">“reqProjectBuildEnded” on page 316</a>

## reqFileListBuildStarted

Description	The IDE sends this request after the user selects the <b>Compile</b> command, and before compiling any files.
Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request prior to compiling files, when the user selects the <b>Compile</b> command.  The IDE sends this request only once, at the start of a <b>Compile</b> operation.
See Also	<a href="#">“Additional Compiler and Linker Requests” on page 223</a> <a href="#">“reqFileListBuildEnded” on page 315</a> <a href="#">“reqProjectBuildStarted” on page 317</a>

## reqLink

Description	Asks a linker plug-in to produce a final output file.
Definition	#include <DropInCompilerLinker.h>

The IDE sends this request when the user chooses **Project > Make**. The IDE uses this request to link the active target's object data into a final library or executable.

**See Also**

[“Linking” on page 213](#)

[“CWGetPluginRequest” on page 130](#)

[“CWDonePluginRequest” on page 109](#)

## **reqMakeParse**

**Description** This flag is reserved.

## **reqPreprocessForDebugger**

**Description** This flag is obsolete. Do not use it.

## **reqPreRun**

**Description** Informs a linker or post-linker that the final target executable is about to be launched.

**Definition** `#include <DropInCompilerLinker.h>`

**Remarks** The IDE issues this request prior to executing the target binary, or any specified helper applications.

**See Also**

[“CWTargetInfo” on page 283](#)

[“CWGetPluginRequest” on page 130](#)

[“CWDonePluginRequest” on page 109](#)

## **reqProjectBuildEnded**

**Description** The IDE sends this request after completing a build operation initiated by the **Make** command.

**Definition** `#include <DropInCompilerLinker.h>`

**Remarks** The IDE sends this request after compiling all files in a project, when the compilation occurred as a result of a **Make** command (rather than a **Compile** command).

The IDE sends this request only once at the end of a **Make** operation.

See Also    [“Additional Compiler and Linker Requests” on page 223](#)  
             [“reqProjectBuildStarted” on page 317](#)  
             [“reqFileListBuildEnded” on page 315](#)

## reqProjectBuildStarted

Description	The IDE sends this request prior to compiling any files when the user selects <b>Make</b> .
Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request before compiling files in response to the user selecting <b>Make</b> .
	The IDE sends this request only once at the start of a <b>Make</b> operation.
See Also	<a href="#">“Additional Compiler and Linker Requests” on page 223</a> <a href="#">“reqProjectBuildEnded” on page 316</a> <a href="#">“reqFileListBuildStarted” on page 315</a>

## reqSubProjectBuildEnded

Description	The IDE sends this request after completing a build of a subproject, when compilation was initiated by the <b>Make</b> command.
Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request at the completion of a subproject <b>Make</b> operation.
	The IDE sends one pair of reqSubProjectBuildStarted / reqSubProjectBuildEnded requests for each subproject encountered during a <b>Make</b> operation.
See Also	<a href="#">“Additional Compiler and Linker Requests” on page 223</a> <a href="#">“reqSubProjectBuildStarted” on page 318</a>

## **reqSubProjectBuildStarted**

**Description** The IDE sends this request prior to compiling a subproject when compilation was initiated by the **Make** command.

**Definition** #include <DropInCompilerLinker.h>

**Remarks** The IDE sends this request when prior to compiling any files in a subproject, during a **Make** operation.

The IDE sends one pair of `reqSubProjectBuildStarted` / `reqSubProjectBuildEnded` requests for each subproject encountered during a **Make** operation.

**See Also** [“Additional Compiler and Linker Requests” on page 223](#)

[“reqSubProjectBuildEnded” on page 317](#)

## **reqTargetBuildEnded**

**Description** The IDE sends this request after compiling a target as part of a **Make** operation.

**Definition** #include <DropInCompilerLinker.h>

**Remarks** The IDE sends this request after compiling all files in a target during a **Make** operation.

The IDE sends one pair of `reqTargetBuildStarted` / `reqTargetBuildEnded` requests for each target encountered during a **Make** operation.

**See Also** [“Additional Compiler and Linker Requests” on page 223](#)

[“reqTargetBuildStarted” on page 318](#)

## **reqTargetBuildStarted**

**Description** The IDE sends this request prior to compiling a target as part of a **Make** operation.

**Definition** #include <DropInCompilerLinker.h>

The IDE sends this request before compiling any files in a target, during a **Make** operation.

The IDE sends one pair of `reqTargetBuildStarted` / `reqTargetBuildEnded` requests for each target encountered during a **Make** operation.

## See Also

[“Additional Compiler and Linker Requests” on page 223](#)

[“reqTargetBuildEnded” on page 318](#)

## reqTargetCompileEnded

## Description

This request is sent when a target’s compile phase ends.

## Definition

```
#include <CompilerMapping.h>
```

## Remarks

This request flag is enabled when the [kCompRequiresTargetCompileStartedMsg](#) flag is set.

## See Also

[“reqTargetCompileStarted” on page 319](#)

## reqTargetCompileStarted

## Description

This request is sent when a target’s compile phase begins.

## Definition

```
#include <CompilerMapping.h>
```

## Remarks

This request flag is enabled when the [kCompRequiresTargetCompileStartedMsg](#) flag is set.

## See Also

[“reqTargetCompileEnded” on page 319](#)

## reqTargetInfo

## Description

Asks a linker plug-in to describe the output it creates.

## Definition

```
#include <DropInCompilerLinker.h>
```

## Remarks

The IDE uses this request to get information about the linker’s output. When it receives this request, the linker should fill out a `CWTTargetInfo` structure and call [CWSetTargetInfo](#).

## See Also

[“Providing Target Information” on page 216](#)

[“CWSetTargetInfo” on page 265](#)

[“CWGetPluginRequest” on page 130](#)

[“CWDonePluginRequest” on page 109](#)

## reqTargetLinkEnded

Description The IDE sends this request after linking a target as part of a **Make** operation.

Definition #include <DropInCompilerLinker.h>

Remarks The IDE sends this request after linking a target, during a **Make** operation.

The IDE sends one pair of reqTargetLinkStarted / reqTargetLinkEnded requests for each target linked during a **Make** operation.

See Also [“Additional Compiler and Linker Requests” on page 223](#)

## reqTargetLinkStarted

Description The IDE sends this request prior to linking a target as part of a **Make** operation.

Definition #include <DropInCompilerLinker.h>

Remarks The IDE sends this request prior to linking a target, during a **Make** operation.

The IDE sends one pair of reqTargetLinkStarted / reqTargetLinkEnded requests for each target linked during a **Make** operation.

See Also [“Additional Compiler and Linker Requests” on page 223](#)

## reqTargetLoaded

Description Tells the plug-in to allocate and set its target storage.

Definition #include <DropInCompilerLinker.h>

Remarks The IDE sends this request when switching targets, to tell the plug-in to allocate its target storage. The IDE sends this request only if the plug-in has set the kCompUsesTargetStorage or linkerUsesTargetStorage flag in its 'Flag' resource.

A plug-in allocates target storage by calling [CWAllocateMemory](#). A plug-in then calls [CWSetTargetStorage](#) to pass the pointer back to the IDE. During subsequent requests from the IDE, a plug-in may call [CWGetTargetStorage](#) to retrieve the stored target data.

**See Also**

- [“CWGetTargetStorage” on page 256](#)
- [“CWSetTargetStorage” on page 266](#)
- [“CWGetPluginRequest” on page 130](#)
- [“CWDonePluginRequest” on page 109](#)

## **reqTargetPrefsChanged**

**Description** Tells the plug-in that the active target’s settings data has changed.

**Definition** #include <DropInCompilerLinker.h>

**Remarks** Upon receiving a `reqTargetPrefsChanged`, a plug-in retrieves its target-related data with [CWGetTargetStorage](#). After updating its data, the plug-in should return the new data to the IDE by calling [CWSetTargetStorage](#). If the plug-in’s target-related data is not dependent on target settings, this request can be ignored.

The IDE sends this request only if the plug-in has set the `kCompUsesTargetStorage` or `linkerUsesTargetStorage` flag in its ‘Flag’ resource.

After completing a request, the plug-in should call [CWDonePluginRequest](#) before returning execution to the IDE.

**See Also**

- [“CWSetTargetStorage” on page 266](#)
- [“CWGetTargetStorage” on page 256](#)
- [“‘Flag’ Resource” on page 645](#)
- [“CWGetPluginRequest” on page 130](#)
- [“CWDonePluginRequest” on page 109](#)

## **reqTargetUnloaded**

**Description**

Tells the plug-in to deallocate its target storage.

Definition	#include <DropInCompilerLinker.h>
Remarks	The IDE sends this request when switching targets, to tell the plug-in to dispose its target storage. The IDE sends this request only if the plug-in has set the <code>kCompUsesTargetStorage</code> or <code>linkerUsesTargetStorage</code> flag in its 'Flag' resource.
	Upon receiving a <code>reqTargetUnloaded</code> request, a plug-in retrieves its target-related data by calling <a href="#">CWGetTargetStorage</a> . If the target-related data was allocated using <a href="#">CWAllocateMemory</a> , the plug-in should call <a href="#">CWFreeMemory</a> to release it. The plug-in should then call <a href="#">CWSetTargetStorage</a> with a NULL target storage pointer to ensure that the stale target storage pointer is not subsequently used by the plug-in.
See Also	<a href="#">“CWGetTargetStorage” on page 256</a> <a href="#">“CWSetTargetStorage” on page 266</a> <a href="#">“'Flag' Resource” on page 645</a> <a href="#">“CWGetPluginRequest” on page 130</a> <a href="#">“CWDonePluginRequest” on page 109</a>

## targetCPU68K

Description	Specifies that the linker generates software for MC680x0 processors.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the <code>targetCPU</code> field of a <a href="#">CWTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for a MC680x0 processor. It may also be returned in the <a href="#">CWTargetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTargetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## **targetCPUAny**

Description     Specifies that the linker generates output that is not processor-specific.

Definition    `#include <DropInCompilerLinker.h>`

Remarks      This constant appears in the targetCPU field of a [CWTTargetInfo](#) structure when calling [CWGetTargetInfo](#) or [CWSetTargetInfo](#) to specify that a linker generates output for any processor. It may also be returned in the [CWTTargetList](#) structure by the [CWPlugin\\_GetDefaultMappingList Entry Point](#).

See Also     [“CWTTargetInfo” on page 283](#)

[“CWGetTargetInfo” on page 255](#)

[“CWSetTargetInfo” on page 265](#)

## **targetCPUEmbeddedPowerPC**

Description     Specifies that the linker generates software for embedded PowerPC processors.

Definition    `#include <DropInCompilerLinker.h>`

Remarks      This constant appears in the targetCPU field of a [CWTTargetInfo](#) structure when calling [CWGetTargetInfo](#) or [CWSetTargetInfo](#) to specify that a linker generates software for embedded PowerPC processors. It may also be returned in the [CWTTargetList](#) structure by the [CWPlugin\\_GetDefaultMappingList Entry Point](#).

See Also     [“CWTTargetInfo” on page 283](#)

[“CWGetTargetInfo” on page 255](#)

[“CWSetTargetInfo” on page 265](#)

## **targetCPUi80x86**

Description     Specifies that the linker generates software for an Intel x86-compatible processor.

Definition    `#include <DropInCompilerLinker.h>`

---

Remarks	This constant appears in the targetCPU field of a <a href="#">CWTTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for an Intel x86-compatible processor. It may also be returned in the <a href="#">CWTargetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTargetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## **targetCPUMips**

Description	Specifies that the linker generates software for a MIPS processor.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for a MIPS processor. It may also be returned in the <a href="#">CWTargetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTargetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## **targetCPUNECv800**

Description	Specifies that the linker generates software for a NEC v800 processor.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for a NEC v800 processor. It may also be returned in the <a href="#">CWTargetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTargetInfo” on page 283</a>

[“CWGetTargetInfo” on page 255](#)

[“CWSetTargetInfo” on page 265](#)

## **targetCPUPowerPC**

Description	Specifies that the linker generates software for a desktop PowerPC processor.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTarqetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for a desktop PowerPC processor. It may also be returned in the <a href="#">CWTarqetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTarqetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## **targetOSAny**

Description	Specifies that the linker generates output that isn't for a specific operating system.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTarqetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates output that isn't for a specific operating system. It may also be returned in the <a href="#">CWTarqetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTarqetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## targetOSEmbeddedABI

Description	Specifies that the linker generates software that conforms to an embedded system's application binary interface.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTarGetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software that conforms to an embedded system's application binary interface. It may also be returned in the <a href="#">CWTarGetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTarGetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## targetOSMacintosh

Description	Specifies that the linker generates software for Apple's Mac OS.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTarGetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for Mac OS. It may also be returned in the <a href="#">CWTarGetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTarGetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## targetOSMagicCap

Description	Specifies that the linker generates software for General Magic's Magic Cap operating system.
Definition	#include <DropInCompilerLinker.h>

Remarks	This constant appears in the targetCPU field of a <a href="#">CWTTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for Magic Cap. It may also be returned in the <a href="#">CWTarGetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTarGetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## targetOSOS9

Description	Specifies that the linker generates software for Microware's OS-9 operating system.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for Microware's OS-9 operating system. It may also be returned in the <a href="#">CWTarGetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .
See Also	<a href="#">“CWTarGetInfo” on page 283</a> <a href="#">“CWGetTargetInfo” on page 255</a> <a href="#">“CWSetTargetInfo” on page 265</a>

## targetOSWindows

Description	Specifies that the linker generates software for the Microsoft Windows operating system.
Definition	#include <DropInCompilerLinker.h>
Remarks	This constant appears in the targetCPU field of a <a href="#">CWTTargetInfo</a> structure when calling <a href="#">CWGetTargetInfo</a> or <a href="#">CWSetTargetInfo</a> to specify that a linker generates software for the Microsoft Windows operating system. It may also be returned in the <a href="#">CWTarGetList</a> structure by the <a href="#">CWPlugin_GetDefaultMappingList Entry Point</a> .

- See Also    [“CWTargetInfo” on page 283](#)  
[“CWGetTargetInfo” on page 255](#)  
[“CWSetTargetInfo” on page 265](#)

## Result Codes for Compiler and Linker Plug-ins

There are two compiler and linker specific error codes:

- [cwErrObjectFileNotStored](#)
- [cwErrUnknownSegment](#)

---

**NOTE** The literal values listed in this section are for reference only (useful during debugging, for example), and do not appear in the headers. Always use constants rather than the literal values in plug-in code.

---

### **cwErrObjectFileNotStored**

Description	No object code exists for this file.
Definition	#include <CWPluginErrors.h> enum { /* ... */ cwErrObjectFileNotStored = 515, /* ... */ };
Remarks	Returned by

### **cwErrUnknownSegment**

Description	The plug-in has specified a segment that doesn't exist.
Definition	#include <CWPluginErrors.h> enum { /* ... */ cwErrUnknownSegment = 513, /* ... */ };

};

Remarks    [CWGetSegmentInfo](#) returns this error when the index supplied is out of range.

## **Compiler and Linker Plug-in Reference**

*cwErrUnknownSegment*

---

# Creating Settings Panel Plug-ins

---

This chapter shows how to create settings panel plug-ins for the CodeWarrior IDE. Settings panels (also known as preference panels) provide user interfaces for configuring other types of plug-ins. This chapter explains general aspects of creating settings panels, and succeeding chapters cover platform-specific details for Windows and Mac OS.

## Creating Settings Panel Plug-ins Overview

Plug-ins that present a user interface to other plug-in settings are known as settings panels (or preference panels). The CodeWarrior IDE displays settings panels in the **Target Settings** dialog for a project.

The IDE supports settings panels on all host platforms. However, there are significant differences in the settings panel portion of the plug-in API, depending upon the host platform on which the IDE is running. This chapter discusses just the aspects of settings panel development that are common to all platforms.

- |         |  |
|---------|--|
| Windows | See <a href="#">“Creating Settings Panel Plug-ins on Windows” on page 375</a> for a discussion of issues specific to Windows settings panel development. |
| Mac OS  | See <a href="#">“Creating Settings Panel Plug-ins on Mac OS” on page 383</a> for a discussion of issues specific to Mac OS settings panel development.   |

The chapter covers the following topics:

- [Platform Differences in Settings Panel API](#)
- [Basic Settings Panel Operation](#)

- [Settings Panel Requests](#)
- [Managing Panel Items](#)

## Platform Differences in Settings Panel API

The services provided by the CodeWarrior IDE for settings panel plug-ins comprise the least uniform portion of the plug-in API across platforms. This reflects CodeWarrior's Mac OS origins, and the evolutionary design of the settings panel API. This section covers general platform differences that pervade the settings panel API.

---

**NOTE**

Future versions of CodeWarrior will eliminate many of the differences between the Windows and Mac OS versions of the panel API.

---

The primary differences between the Windows and Mac OS versions of the API are:

- On Windows, plug-in context is opaque. On Mac OS, direct access to plug-in state is required to implement a settings panel. plug-in state is passed to the plug-in's main entry point as an opaque context parameter of type [CWPluginContext](#) on Windows, and as a parameter of type [PanelParameterBlock](#) on Mac OS.
- On Windows, some basic user interface functionality is taken care of automatically by the IDE. On Mac OS, the plug-in must instead implement this functionality (for example, for "drag and drop" related operations for custom items).
- There are a small number of API functions provided only on Mac OS, to support Mac OS-specific functionality. Specifically, functions for handling AppleEvents are not available on Windows.
- Most panel routines are common to both platforms. However, their names and parameter types differ in certain uniform ways. See "[Differences in Common Panel Routines](#)" for more details.
- The Mac OS API includes some deprecated routines and IDE requests for implementing custom controls, applicable only to panel interfaces constructed using 'DITL's.

This chapter focuses on aspects of settings panels that are common across platforms. However, as a consequence of the way plug-in context is maintained on Windows and Mac OS, in some cases it is necessary to discuss differences in the way common functionality is implemented. Also, most of the code listings use Windows API function names, but in general, converting them to Mac OS is straightforward.

## **Settings Panel Context**

The main entry points for Windows and Mac OS panel plug-ins are passed different parameters. Both provide the context, or state, of a plug-in. In other parts of the API, this context is fully opaque, accessible through only various API routines. In the settings panel API on Mac OS, however, the settings panel state is maintained in a publicly accessible parameter block passed to the plug-in instead of as opaque context.

[Listing 8.2](#) illustrates the declaration of a Windows settings panel main entry point.

---

### **Listing 8.1 Windows settings panel plug-in main entry point**

---

```
CWPLUGIN_ENTRY (main) (CWPluginContext context)
```

---

[Listing 8.2](#) illustrates the declaration of a Mac OS settings panel main entry point.

---

### **Listing 8.2 Mac OS settings panel plug-in main entry point**

---

```
CWPLUGIN_ENTRY (main) (PanelParameterBlock *pb)
```

---

The difference in the context parameter requires that Mac OS plug-ins access specific fields of the settings panel parameter block directly, whereas Windows plug-ins call the IDE to obtain the same information. All of the following state accessor calls are Windows-specific:

- [CWPanelGetCurrentPrefs](#)
- [CWPanelGetOriginalPrefs](#)
- [CWPanelGetFactoryPrefs](#)

## Creating Settings Panel Plug-ins

### Settings Panel Context

---

- [CWPanelGetDebugFlag](#)
- [CWPanelSetRevertFlag](#)
- [CWPanelSetFactoryFlag](#)
- [CWPanelSetResetPathsFlag](#)
- [CWPanelSetRecompileFlag](#)
- [CWPanelSetRelinkFlag](#)
- [CWPanelSetReparseFlag](#)

To implement the functionality provided by these routines on Mac OS, a plug-in directly examines and modifies fields in the [PanelParameterBlock](#) structure passed to the plug-in. This parameter block has the following declaration:

#### **Listing 8.3 PanelParameterBlock structure**

```
/* parameter block -- this is passed to the dropin at each request */
typedef struct PanelParameterBlock {
    /* common to all dropins */
    long    request;      /* [->] requested action (see below)      */
    long    version;      /* [->] version # of shell's API          */
    void   *context;      /* [->] reserved for use by shell          */
    void   *storage;      /* [<->] reserved for use by the dropin */
    FSSpec targetfile;   /* [->] FSSpec of current project        */

    /* specific to panels */
    CWDialo g dialog;
    /* [->] pointer to PreferencesÉ dialog      */
    Handle   originalPrefs;
    /* [->] panel's original options data       */
    Handle   currentPrefs;
    /* [<->] panel's current options data       */
    Handle   factoryPrefs;
    /* [<->] panel's "factory" options data     */
    EventRecord *event;
    /* [->] dialog event (for reqFilterEvent) */
    short    baseItems;
    /* [->] # of items in dialog shell          */
    short    itemHit;
    /* [<->] for reqFilterEvent and reqItemHit */
}
```

```
Boolean      canRevert;
/* [<-] enable Revert button */ 
Boolean      canFactory;
/* [<-] enable Factory button */ 
Boolean      reset;
/* [<-] access paths must be reset */ 
Boolean      recompile;
/* [<-] files must be recompiled */ 
Boolean      relink;
/* [<-] project must be relinked */ 
AEKeyword    prefsKeyword;
/* [->] for reqAEGetPref and reqAESetPref */
AEDesc      prefsDesc;
/* [->] for reqAESetPref */ 
Boolean      debugOn;
/* [->] turning on debugging? */ 
FSSpec       oldtargfile;
/* [->] previous project file FSSpec */ 

/* version 2 API */
CWPanelCallbacks*      callbacks;

/* version 3 API */
Boolean      reparse;           /* [<-] project must be reparsed */ 

/* version 4 API */
DragReference   dragref;      /* [->] for drag-related requests */ 
Rect          dragrect;     /* [<-] rect to track mouse in */ 
Point         dragmouse;    /* [->] mouse location during drag */ 

/* version 5 API */
unsigned char   toEndian;    /* [->] for reqByteSwapData, the
endian we are swapping to */

/* CWPro 3 temporary placeholders for opaque references to prefs
data.
These will be removed in Pro 4.
*/
CWMemHandle   originalPrefsMemHandle;
CWMemHandle   currentPrefsMemHandle;
CWMemHandle   factoryPrefsMemHandle;
CWMemHandle   panelPrefsMemHandle;
```

## Creating Settings Panel Plug-ins

### Settings Panel Context

---

```
/* version 11 api */
long listViewCellRow; /* [->] the cell row of the listView */
long listViewCellCol; /* [->] the cell row of the listView */
/*
These fields are supported only from DropinPanelPrefVersion 12
and higher for long filename support. Older plug-ins will get
will use FSSpec based fields targetFile and oldTargetFile.
Since the IDE is LFN enabled, the will attempt to convert the
internal LFN data struture to create a valid FSSpec. If the
API fails to create a valid FSSpec, the FSSpec name field
would be filled with the project name.

The newer plug-ins or those plug-ins that decide to upgrade to
the newer library(PluginLib5) will have to be aware that these
fields are the new means of accessing the target file and old
target file. The older fsspec based fields will be deprecated
for the newer plug-ins (version 12 and higher) and will be
zeroed out.
*/
/* version 12 api */
CWPanelFileSpec lfnTargetFile;
CWPanelFileSpec lfnOldTargetFile;

} PanelParameterBlock, *PanelParameterBlockPtr;
```

---

Not all fields of the [PanelParameterBlock](#) structure should be modified by plug-ins. Fields sent to the plug-in by the IDE are marked with ' ->' in the comments. Similarly, fields that can be modified by a plug-in are marked with ' <- '.

To access a plug-in's current settings data, for example, a plug-in would directly refer to the `originalPrefsMemHandle` field of the [PanelParameterBlock](#). Note that the handles used in this parameter block are traditional Mac OS handles, not CodeWarrior [CWMemHandle](#)s. Also note that the plug-in should not modify the handle itself, but rather the contents of the handle.

Where appropriate, the rest of this chapter will indicate other fields that can be modified by a plug-in. See [CWMemHandle](#) for more information.

**NOTE** Because the Windows portion of the settings API is simpler, most of the code examples presented in this chapter contain only Windows code. Modifying them to work on Mac OS is usually straightforward.

---

## Differences in Common Panel Routines

Because the context passed to settings panel plug-ins differs on Windows and Mac OS, the declarations for all settings panel API routines differ across platforms. With the exception of differences documented in “[Creating Settings Panel Plug-ins on Windows](#)” and “[Creating Settings Panel Plug-ins on Mac OS](#),” the same set of routines is provided on both platforms. Common routines are functionally identical but named differently, though consistently.

The differences between the Windows and Mac OS API routine declarations consist of the following:

- On Windows, the first parameter to all routines is of type [CWPluginContext](#). This parameter is opaque state, private to the IDE. On Mac OS, the first parameter to all routines is of type [PanelParamBlkPtr](#). This parameter points to a publicly declared structure accessed directly by plug-ins, as explained in [Settings Panel Context](#).
- Where Windows routines require parameters of type [CWMemHandle](#), Mac OS routines require parameters of type Handle. Handle is a Mac OS toolbox type defined in the Mac OS system headers. Use the Mac OS system routines HLock and HUnlock to lock and unlock Handles, rather than [CWLockMemHandle](#) and [CWUnlockMemHandle](#).
- Windows panel routines use C strings, where Mac OS panel routines use Pascal strings instead.
- On Windows, all settings panel routines except those for XML import and export start with a ‘CWPanel’ prefix. On Mac OS, all settings panel routines start with a ‘CWPanl’ prefix. Windows XML import and export routines start with just a ‘CW’ prefix.

Throughout this chapter, all routine declarations are listed using the Windows declarations. The corresponding Mac OS declaration can easily be inferred, by substituting prefixes and handle types, as outlined above.

# Basic Settings Panel Operation

This section describes the operation and general responsibilities of settings panels. It covers the entry points exported by a plug-in and discusses settings panel data.

## Settings Panel Overview

A settings panel is the user-visible front end to internal binary data used to configure other plug-ins. Settings panels allow the user of the IDE to configure the operation of a plug-in, such as a compiler, linker, or version control plug-in. Settings panels are displayed in the **Edit > Target Settings** and **Edit > Version Control Settings** dialog boxes.

The basic responsibilities of a settings panel plug-in consist of:

1. presenting and operating a user interface
2. extracting data values from the user interface, and installing data values into the user interface
3. maintaining settings data

As with other CodeWarrior plug-ins, settings panels are implemented as loadable code libraries that export standard entry points, most of which provide information about the plug-in to the IDE. In addition, settings panel typically include resource descriptions of the user interface items that install in the IDE's settings windows.

Like other plug-ins, settings panel plug-ins export a main entry point that handles requests from the IDE. Most of a settings panel's functionality is implemented as code that responds to requests from the IDE. plug-in main entry points and requests sent by the IDE to all plug-ins are discussed in "["Responding to IDE Requests."](#)" Specific settings panel request codes are discussed in "["Settings Panel Requests."](#)"

## Settings Panel Entry Points

Settings panels are packaged as dynamically loaded libraries, like all other IDE plug-ins. For more information about settings panel binary formats, see "["Plug-in Binary Formats and Installation."](#)"

Settings panel plug-ins implement the following standard entry points:

1. An entry point for reporting plug-in capabilities, `CWPlugin_GetDropInFlags`. See [“Specifying Plug-in Capabilities” on page 65](#) for more information.
2. An entry point that reports the plug-in’s internal name, `CWPlugin_GetDropInName`. See [“Specifying a Plug-in’s Dropin and Display Names” on page 70](#) for more information.
3. An entry point that reports the plug-in’s localized, user-visible display name, `CWPlugin_GetDisplayName`. See [“Specifying a Plug-in’s Dropin and Display Names” on page 70](#) for more information.
4. An optional entry point that reports the panel’s family name and type. Family names and types are used to group related settings panels in the **Target Settings** dialog. Most settings panels do not need to implement this entry point. See [“Specifying Associated Settings Panels” on page 72](#) for more information.
5. An optional entry point which provides the name of an associated help file to the IDE, `CWPlugin_GetHelpInfo`. This is only supported on Windows. See [Help Information Entry Point](#) for more information.
6. A main entry point that handles [Settings Panel Requests](#). See [“Responding to IDE Requests” on page 75](#) for more information about basic request handling.

Unlike the other entry points common to other plug-in types, the `CWPlugin_GetDropInFlags` entry point is implemented somewhat differently in settings panels. Specifically, settings panel plug-ins return a [PanelFlags](#) structure, rather than a [DropInFlags](#) structure.

A [PanelFlags](#) structure differs from a [DropInFlags](#) structure in some of the fields returned. As explained in [“Specifying Plug-in Capabilities.”](#) a settings panels plug-in must cast the returned [PanelFlags](#) pointer to a [DropInFlags](#) pointer prior to return, due to the way the `CWPlugin_GetDropInFlags` entry point is declared. For an explanation of the fields in a [PanelFlags](#) structure, see [PanelFlags Structure](#).

The dropinflags field of a `PanelFlags` structure has different meaning than it does in a `DropInFlags` structure. See [Settings Panel Flags](#) for the meanings of individual flags.

## PanelFlags Structure

The `CWPlugin_GetDropInFlags` entry point for settings panel plug-ins has the following declaration:

---

```
CWPLUGIN_ENTRY (CWPlugin_GetDropInFlags)
    (const DropInFlags** flags, long* flagsSize);
```

---

This entry point returns information to the IDE about the plug-in's capabilities in a `PanelFlags` structure. See [“Specifying Preference Panel Capabilities” on page 68](#) for a typical implementation and more information about the `CWPlugin_GetDropInFlags` entry point.

Although the entry point is declared to return a structure of type `DropInFlags`, in fact settings panels return a structure of type `PanelFlags`. This structure contains the following fields:

---

Field name:	Meaning:
rsrcversion	The version of the <code>PanelFlags</code> structure. Use the number 3 for the latest version. Future versions of the headers will include a define for this value.
dropintype	For panels, this should be set to 'PanL'.
earliestCompatibleAPIVersion	The earliest version of the settings panel API this plug-in knows how to operate with. Specify this using one of the <code>DROPINPANELAPIVERSION</code> defines.
dropinflags	Specifies settings panel capabilities. See <a href="#">Settings Panel Flags</a> for details.

---

<b>Field name:</b>	<b>Meaning:</b>
panelfamily	Lists the family (heading) this panel will be grouped under in the <b>Target Settings</b> dialog. See <a href="#">Specifying Panel Family</a> for more information.
newestAPIVersion	The latest version of the settings panel API this plug-in knows how to operate with. Usually, you will want to specify <code>DROPINPANELAPIVERSION</code> for this field.
dataversion	Specifies the version of the settings data created and returned by this plug-in. This value is determined exclusively by the plug-in but should match the first 16-bit integer stored in the plug-in's setting data.
panelscope	Specifies where the settings panel will appear in the IDE user interface. Use one of the following values: <a href="#">panelScopeGlobal</a> , <a href="#">panelScopeProject</a> , <a href="#">panelScopeTarget</a> . Most plug-ins use <code>panelScopeTarget</code> .

## Settings Panel Flags

The value of `dropinflags` returned by settings panel plug-ins consists of the bitwise-or (or arithmetic sum) of the following constants:

## Creating Settings Panel Plug-ins

### Settings Panel Data

---

Flag name	Meaning
usesStrictAPI	Applies to Mac OS plug-ins only. All plug-ins should set this flag; future versions of the IDE will not support plug-ins compiled with usesStrictAPI set to false.
	Indicates that the panel uses only plug-in API routines to manipulate its panel UI elements. When clear, a plug-in may use Mac OS API calls (ex.: GetDialogItem) to manipulate dialog items.
supportsByteSwapping	Indicates that the panel supports the <a href="#">reqByteSwapData Request</a> , used to convert settings data between big and little “endian” formats. See <a href="#">reqByteSwapData</a> for more information
supportsTextSettings	Indicates that the panel supports the <a href="#">reqReadSettings</a> and <a href="#">reqWriteSettings</a> requests, used to import and export settings data as XML. For more information, see <a href="#">reqReadSettings Request</a> and <a href="#">reqWriteSettings Request</a> .

---

**NOTE** All unused flags should be set to zero, and are reserved by Metrowerks for future use.

---

## Settings Panel Data

Settings panels are responsible for maintaining internal binary data that corresponds to user-settable interface elements. This data is stored with a project by the IDE, and returned to plug-ins when

necessary, either as the result of an API call on Windows, or in the [PanelParameterBlock](#) on Mac OS.

### Modifying Settings Data

To modify the settings data maintained for the plug-in by the IDE, a plug-in simply changes the values stored in the appropriate IDE handle directly. For example, the IDE stores a handle containing factory defaults for a plug-in. The values of this data are specified by a plug-in, but maintained by the IDE. When the IDE asks the plug-in to return its factory defaults (by sending a [reqGetFactoryRequest](#)), the plug-in obtains the current defaults handle, and directly modifies its contents. The changes are made to the IDE's copy of the data, and no additional API calls are required to effect the changes.

To access data stored in settings handles, a plug-in can use either of two techniques:

1. Lock the handle using [CWLockMemHandle](#) on Windows or [HLock](#) on Mac OS; access and modify fields in the handle directly; and unlock the handle when finished, using [CWUnlockMemHandle](#) on Windows or [HUnlock](#) on Mac OS. This is the technique illustrated in the Windows sample code.
2. Copy data from the handle into a local structure, access and modify it, and when finished, copy it back into the handle. This is the technique illustrated in the Mac OS sample code.

When storing data in a handle, plug-ins may need to resize the handle first. This will typically be necessary, for example, when handling a [reqGetFactory Request](#) for the first time, or when handling a [reqGetData Request](#) when a plug-in's data varies in size (this often happens, for example, when a plug-in's data includes text).

Windows

On Windows, the IDE stores settings data in memory blocks of type [CWMemHandle](#). When a plug-in requires access to its current settings data, it calls [CWPanelGetCurrentPrefs](#). To obtain the most recent settings data prior to user modifications in the settings panels, the plug-in calls [CWPanelGetOriginalPrefs](#). To obtain the factory defaults maintained for the current panel, a plug-in calls [CWPanelGetFactoryPrefs](#).

## **Creating Settings Panel Plug-ins**

### *Settings Panel Data*

---

These calls return a [CWMemHandle](#) which can be locked using [CWLockMemHandle](#) prior to accessing or modifying its contents, and unlocked using [CWUnlockMemHandle](#). A plug-in may change the size of its settings data handle using [CWResizeMemHandle](#).

Mac OS On Mac OS, the IDE passes a [PanelParameterBlock](#) to the plug-in, which contains the plug-in's current settings data in the `currentPrefs` handle. The `originalPrefs` and `factoryPrefs` fields contain the most recently committed settings data, and the current factory defaults data, respectively.

These fields provide the settings data in regular Mac OS Handles, not [CWMemHandles](#). Mac OS handles can be locked and unlocked using `HLock` and `HUnlock`, and resized using `GetHandleSize` and `SetHandleSize`.

### **Settings Data Versioning**

With one exception, the IDE makes no assumptions about the contents of a plug-in's data. The values and the types of the data a panel plug-in stores in its preferences handle are entirely up to the plug-in - except for the first two bytes.

The IDE assumes that the first 16-bit integer of a plug-in's data specifies the integer version of the plug-in data. plug-ins should generally start numbering version data at one (1), and increment the version number by one for each plug-in release that changes the settings data layout, either by adding new values, removing old values, or changing the interpretation of existing values.

The IDE uses the version number to determine when to send plug-ins the [reqUpdatePref](#) message. When the version number appearing in settings data loaded from a project is less than the version number specified by a panel in the `dataVersion` field of its [PanelFlags](#), the IDE sends the plug-in an [reqUpdatePref](#) request.

Plug-ins can either update the old settings data to the new format, or inform the IDE that the data is too old, by returning an error. For more information, see [“reqUpdatePref Request” on page 362](#).

# Settings Panel Requests

This section discusses the requests sent by the IDE to settings panel plug-ins, and outlines how to handle each request.

A properly implemented settings panel responds to a relatively large number of requests. These are discussed in the following sections:

- [Handling Settings Panel Requests](#)
- [Determining the “Hit” Dialog Item](#)
- [Returning Errors to the IDE](#)
- [Initialization and Shutdown](#)
- [Getting and Putting Settings Data](#)
- [Manipulating Settings Data](#)
- [Handling User Interaction](#)
- [Importing and Exporting Settings Data](#)

## Handling Settings Panel Requests

Settings panels receive requests from the IDE in the same way other plug-ins do. The IDE sends a request code to the plug-in’s main entry point, indicating the action to be taken by a plug-in.

The request code is sent differently on Window and Mac OS. On Windows, a plug-in calls [CWGetPluginRequest](#). On Mac OS, the plug-in inspects the `request` field of the [PanelParameterBlock](#).

The value and meaning of the requests codes are the same for both platforms. Some requests are not sent to Windows plug-ins, however (for example, drag and drop requests, which are handled automatically, and AppleEvent requests, which apply only to Mac OS).

[Listing 8.4](#) illustrates a typical settings panel main entry point, consisting of a large switch statement that determines the current plug-in request and calls the appropriate subroutine. Note that the latter portion of the switch statement handles requests that are only relevant on Mac OS. Also note the conditional code for retrieving the plug-in request, and for returning an error result.

**Listing 8.4 Settings panel main entry point request handler**

```
CWPLUGIN_ENTRY(main) (CWPluginContext context)
{
    CWResult    result = cwNoErr;
    long        request;

#if Windows
    result = CWGetPluginRequest(context, &request);
    if (result != cwNoErr)
        return result;
#else /* Macintosh */
    request = pb->request;
#endif

    // Dispatch on compiler request
    switch (request)
    {
        case reqInitialize:
            /* panel has just been loaded into memory */
            break;

        case reqTerminate:
            /* panel is about to be unloaded from memory */
            break;

        case reqInitDialog:
            /* add our controls to the preferences dialog */
            result = InitDialog(context);
            break;

        case reqTermDialog:
            /* remove our controls from the preferences dialog */
            TermDialog(context);
            break;

        case reqPutData:
            /* put current settings into our dialog items */
            result = PutData(context);
            break;

        case reqGetData:
```

```
/* get current settings from our dialog items */
result = GetData(context, TRUE);
break;

case reqByteSwapData:
    /* byte swap the data in the handle */
    result = ByteSwapData(context);
    break;

case reqItemHit:
    /* handle a hit on one of our dialog items */
    result = ItemHit(context);
    break;

case reqValidate:
    /* determine if we need to reset paths, recompile,
       relink, or reparse */
    result = Validate(context);
    break;

case reqGetFactory:
    /* return our factory settings */
    result = GetFactory(context);
    break;

case reqUpdatePref:
    /* update our settings data to the latest version */
    result = UpdatePref(context);
    break;

case reqReadSettings:
    /* read settings data values from XML */
    result = ReadSettings(context);
    break;

case reqWriteSettings:
    /* write settings data values to XML */
    result = WriteSettings(context);
    break;

#endif Macintosh /* Macintosh-specific requests */
case reqFilter:
```

## **Creating Settings Panel Plug-ins**

### *Handling Settings Panel Requests*

---

```
/* filter an event in the dialog */
result = Filter(pb, pb->event, &pb->itemHit);
break;

case reqDrawCustomItem:
    /* draw one of our user items (CW8 and later) */
    DrawCustomItem (pb);
    break;

case reqActivateItem:
    /* switch input focus to a custom item, higlighting
       it if necessary */
    ActivateCustomItem (pb, true);
    break;

case reqDeactivateItem:
    /* switch input focus away from a custom item,
       unhiglighting it if necessary */
    ActivateCustomItem (pb, false);
    break;

case reqHandleKey:
    /* detect and handle any special key combinations */
    HandleKey(pb, pb->event);
    break;

case reqHandleClick:
    /* detect and handle mouse clicks (usually only
       needed in custom items) */
    HandleClick(pb, pb->event);
    break;

case reqFindStatus:
    /* return enabled/disabled status for edit menu items */
    FindStatus(pb);
    break;

case reqObeyCommand:
    /* handle an edit menu command */
    ObeyCommand(pb);
    break;
```

```
case reqAEGetPref:
    /* return one settings data item as an Apple Event
       descriptor */
    result = GetPref(pb->prefsKeyword, &pb->prefsDesc,
                     pb->currentPrefs);
    break;

case reqAESetPref:
    /* change one settings data item to the value given
       by an Apple Event descriptor */
    result = SetPref(pb->prefsKeyword, &pb->prefsDesc,
                     pb->currentPrefs);
    break;

case reqDragEnter:
    /* determine if we can accept the drag and, if so,
       start tracking */
    result = DragEnter(pb);
    break;

case reqDragWithin:
    /* continue tracking */
    DragWithin(pb);
    break;

case reqDragExit:
    /* stop tracking */
    DragExit(pb);
    break;

case reqDragDrop:
    /* the user has dropped in our panel */
    DragDrop(pb);
    break;

default:
    result = paramErr;
    break;
}

#endif

default:
```

## **Creating Settings Panel Plug-ins**

### *Determining the “Hit” Dialog Item*

---

```
    /* unfamiliar request */
    result = cwErrRequestFailed;
    break;
}

/* Return result code */
#ifndef Windows
    result = CWDonePluginRequest(context, result);
#endif
return (result);
}
```

---

## **Determining the “Hit” Dialog Item**

Many panel requests pertain to a specific dialog item. This item is referred to as the “hit” item (because often it is the subject of user interaction, with mouse or keyboard).

Plug-ins determine the hit item by first obtaining the number of the hit item, and then subtracting the “base” number of items in the dialog. The base number of item reflects the number of items in the IDE’s panel dialog, prior to addition of the plug-in’s items.

- |         |  |
|---------|--|
| Windows | On Windows, a plug-in determines the base number of panel items by calling <a href="#">CWPanelGetNumBaseDialogItems</a> . A plug-in determines the hit item by calling <a href="#">CWPanelGetDialogItemHit</a> . |
|---------|--|

[Listing 8.5](#) shows how to determine the true hit item on Windows.

### **Listing 8.5 Determining the hit item on Windows**

---

```
short      numBaseItems;
short      theItem;

CWPanelGetNumBaseDialogItems(context, &numBaseItems);
CWPanelGetDialogItemHit(context, &theItem);
/* compute true hit item number: */
theItem -= numBaseItems;
```

---

Mac OS      On Mac OS, the IDE passes the base number of panel items in the `baseItems` field of the [PanelParameterBlock](#). The IDE passes the hit item number in the `itemHit` field.

[Listing 8.6](#) shows how to determine the true hit item on Mac OS.

### **Listing 8.6   Determining the hit item on Mac OS**

---

```
short           theItem;

/* compute true hit item number: */
theItem = pb->itemHit - pb->baseItems;
/* pb is a pointer to the panel parameter block */
```

---

**NOTE**      On both Windows and Mac OS, the number of base dialog items will be zero. The number was only nonzero for older versions of the IDE, supporting 'DITL'-based panels on Mac OS. Current Mac OS plug-ins should use 'PPob' resources to construct their user interfaces (see [User Interface Resources](#)). However, panels should still follow this procedure for determining the hit item.

---

## **Returning Errors to the IDE**

After servicing a request, and immediately before returning execution to the IDE, a Windows plug-in returns an error code to the IDE by calling [CWDonePluginRequest](#). The plug-in should then return the result of this call to the IDE, as its `main()` function result as described in [“Reporting Errors to the IDE” on page 98](#).

Mac OS      Because Mac OS plug-ins are not sent a context parameter, Mac OS plug-ins cannot obey standard protocol for returning errors. Instead, Mac OS settings panel plug-ins simply return an error code to the IDE as the result of their `main()` function.

[Listing 8.4](#) illustrates the proper way to return error codes on both Windows and Mac OS.

### **Additional Panel Error Codes**

Settings panels, like other plug-ins, may return any of the error codes listed in `CWPluginErrors.h`. A plug-in reports that it has

successfully handled a request by returning `cwNoErr`. If a plug-in returns any other value, the IDE attempts to recover, and reports an error to the user, if necessary.

In addition to the standard error codes, the IDE reports and recognizes certain additional panel error codes. These error codes are defined in `DropInPanelWin32.h` (for Windows) and `DropInPanel.h` (for Mac OS) and have the following meanings:

**Table 8.1** **Panel Error Codes:**

<b>Error code:</b>	<b>Meaning:</b>
<code>kBadPrefVersion</code>	A plug-in returns this result to the IDE to indicate that it does not understand this version of its plug-in data. This happens when later versions of plug-ins drop support for old settings data formats, or when newer plug-in data is opened with an older plug-in.
<code>kMissingPrefErr</code>	Not currently used.
<code>kSettingNotFoundErr</code>	Returned when a plug-in calls one of the <code>CWRead</code> calls or <a href="#"><code>CWGetNamedSetting</code></a> and an XML setting by the specified name cannot be found.
<code>kSettingTypeMismatchErr</code>	Returned when a plug-in requests a named setting whose type does not match the type implied by the routine requesting the setting. For example, this error would be returned if a plug-in requested a boolean setting using <a href="#"><code>CWReadStringSetting</code></a> .
<code>kInvalidCallbackErr</code>	

<b>Error code:</b>	<b>Meaning:</b>
	Returned when the plug-in calls an API routine that is not available during the current request. For example, the IDE returns this result if a plug-in calls an XML routine during a request other than <a href="#">reqReadSettings</a> or <a href="#">reqWriteSettings</a> .
kSettingOutOfRangeErr	Not currently used; may be used in the future.

## Initialization and Shutdown

The IDE notifies settings panel plug-ins when it loads them, unloads them, displays their panels, and when it removes them from the **Target Settings** dialog. In response, plug-ins should initialize their state, clean up their state, and construct and dispose their user interfaces, respectively.

### `reqInitialize (reqInitPanel) Request`

Panel plug-ins should handle the [reqInitialize](#) request in the same way other plug-ins handle the request, by initializing internal state, and acquiring any necessary operating system resources. For more information about how to handle the [reqInitialize](#) request, see [reqInitialize Request](#).

---

**NOTE** The [reqInitPanel](#) enumerated constant is a deprecated synonym for [reqInitialize](#).

---

### `reqTerminate (reqTermPanel) Request`

Panel plug-ins should handle the [reqTerminate](#) request in the same way other plug-ins handle the request, by cleaning up internal state, and releasing any necessary operating system resources. For more information about how to handle the [reqTerminate](#) request, see [reqTerminate Request](#).

## **Creating Settings Panel Plug-ins**

### *Initialization and Shutdown*

---

- NOTE** The [reqTermPanel](#) enumerated constant is a deprecated synonym for [reqTerminate](#).
- 

#### **reqInitDialog Request**

When the IDE sends a [reqInitDialog](#) request, a settings panel builds its user interface by adding items to the existing preferences dialog. This is usually as simple as appending dialog items, loaded from standard dialog resources (on Windows) or PowerPlant 'PPop' resources (on Mac OS), to the preferences dialog using [CWPanelAppendItems](#), and setting the initial state of the items. User interface controls may be enabled and disabled using [CWPanelEnableItem](#), and control values may be initialized using [CWPanelSetItemValue](#) and [CWpanelSetItemText](#).

- NOTE** Normally, little or no control initialization is required in response to a [reqInitDialog](#) request. This is because the IDE sends the [reqPutData](#) request before displaying a panel, giving the plug-in a chance to initialize its user interface controls.
- 

For certain types of user interface controls, plug-ins must perform additional initialization. For example, the IDE does not support creating the items of a Windows list box from a resource description. A plug-in must manually add items to the list box during panel initialization.

[Listing 8.7](#) illustrates how to handle a [reqInitDialog](#) request.

#### **Listing 8.7 Handling a reqInitDialog request**

---

```
CWResult InitDialog(CWPluginContext context)
{
    CWResult result = cwNoErr;

    result = CWPanelAppendItems(context, kItemListID);
    if (result == cwNoErr)
    {
        /* Add items to a combo box: */
        CWPanelInsertListItem(context, kComboBoxItem, 1, "Item 1");
        CWPanelInsertListItem(context, kComboBoxItem, 2, "Item 2");
        CWPanelInsertListItem(context, kComboBoxItem, 3, "Item 3");
```

---

```
/* Set initial value of checkbox and enable it: */
CWPanelSetItemValue (context, kCheckboxItem, 1);
CWPanelEnableItem (context, kCheckboxItem, 1);
}
return (result);
}
```

---

#### **reqTermDialog Request**

Settings panel plug-ins receive `reqTermDialog` requests to shut down their user interfaces. In most cases, little action is necessary since the IDE disposes user interface items automatically. However, a plug-in should be sure to free any other items acquired when responding to a [reqInitDialog](#) request, such as OS resources.

#### **reqFirstLoad Request**

The IDE sends the [reqFirstLoad](#) request to all panels when a target is first loaded. This normally occurs when switching targets or opening projects. Most plug-ins do not need to respond to this request.

## **Getting and Putting Settings Data**

When requested by the IDE, plug-ins should install internal binary settings data into their user interface controls, and extract settings values from user interface controls and return them to the IDE.

#### **reqPutData Request**

The IDE sends the [reqPutData](#) request when it displays the plug-in panel's UI. The [reqPutData](#) request tells the plug-in to copy internal binary data values stored in the current settings handle into the panel's UI elements, for display to and modification by the user.

[Listing 8.8](#) shows how to respond to a [reqPutData](#) request.

#### **Listing 8.8 Handling a reqPutData request**

---

```
CWResult PutData(CWPluginContext context)
{
```

---

## Creating Settings Panel Plug-ins

### Getting and Putting Settings Data

---

```
CWResult      result = cwNoErr;
CWMemHandle   memHandle = NULL;
SamplePref     *prefsPtr;

/* Get a pointer to the current prefs */
result = CWPanelGetCurrentPrefs(context, &memHandle);
if (result == cwNoErr)
    result = CWLockMemHandle(context, memHandle,
                           FALSE, (void**)&prefsPtr);

/* Stuff data into the preference dialog */
if (result == cwNoErr)
{
    CWPanelSetItemValue(context, kLinkSymItem,
                        prefsPtr->linksym);
    /* add 1 to item, because popups are 1-based */
    CWPanelSetItemValue(context, kProjTypeItem,
                        prefsPtr->projtype + 1);
    CWPanelSetItemText (context, kOutFileItem,
                        prefsPtr->outfile);

    /* Relinquish our pointer to the prefs data */
    if (memHandle)
        /* note: errors here intentionally ignored */
        CWUnlockMemHandle(context, memHandle);
}

return (result);
}
```

---

### **reqGetData Request**

The IDE sends the [reqGetData](#) request when saving settings data established by the user in the plug-in's settings panel. The [reqGetData](#) request tells the plug-in to extract settings values from the panel's UI, and to store them in the plug-in's current preferences data handle.

[Listing 8.9](#) shows how to respond to a [reqGetData](#) request.

**Listing 8.9 Handling a reqGetData request**

```
CWResult GetData(CWPluginContext context)
{
    CWResult      result = cwNoErr;
    CWMemHandle  memHandle = NULL;
    SamplePref   *prefsPtr;
    SamplePref   prefsCopy;
    long         symval;
    long         projval;

    /* Get a pointer to the current prefs */
    result = CWPanelGetCurrentPrefs(context, &memHandle);
    if (result == cwNoErr)
        result = CWLockMemHandle(context, memHandle,
                                  FALSE, (void**)&prefsPtr);

    if (result == cwNoErr)
    {
        /* Work on a copy of the prefs data in case there
           is an error along the way */
        prefsCopy = *prefsPtr;

        /* Stuff dialog values into the current prefs */
        result = CWPanelGetValue(context, kLinkSymItem,
                               &symval);
        if (result == cwNoErr)
            result = CWPanelGetValue(context, kProjTypeItem,
                               &projval);
        prefsCopy.linksym = symval;
        /* subtract 1 from item, because popups are 1-based */
        prefsCopy.projtype = projval - 1;
        if (result == cwNoErr)
            result = CWPanelGetItemText(context, kOutFileItem,
                               prefsCopy.outfile,
                               sizeof(prefsCopy.outfile));

        if (result == cwNoErr)
            /* if successful, update the real prefs data */
            *prefsPtr = prefsCopy;
    }
}
```

```
/* Relinquish our pointer to the prefs data */
if (memHandle)
    // note: errors here intentionally ignored
    CWUnlockMemHandle(context, memHandle);

return result;
}
```

---

## Manipulating Settings Data

The IDE issues numerous requests for modifying settings data. Since the IDE does not know anything about the format of a plug-in's data (except its version), it sends requests to plug-ins to get default settings, update older preference data, change byte ordering, and notify plug-ins of project configuration changes. The requests apply to the current settings data maintained by the IDE.

### **reqGetFactory Request**

The IDE sends the [reqGetFactory](#) request to tell the plug-in to return default values for its preference data. A plug-in should get the current factory settings handle, resize it if necessary, and place default values for all settings in the handle.

---

**NOTE** The IDE also sends the [reqGetFactory](#) request when attempting to open newer settings than those supported by a panel plug-in.

---

All panel plug-ins must store a version number in the first two bytes of their settings handle. This is the only restriction the IDE places on a panel's settings data. The IDE uses the version number to determine when to update settings data loaded from older projects. See [Settings Data Versioning](#) for more information.

[Listing 8.10](#) illustrates this process.

---

### **Listing 8.10 Handling a reqGetFactory request**

---

```
CWResult GetFactory(CWPluginContext context)
{
    CWResult result = cwNoErr;
```

---

```
CWMemHandle memHandle = NULL;
SamplePref *factoryPrefs;

/* Retrieve the current factory settings
   (reminder: handle may have length of zero) */
result = CWPanelGetFactoryPrefs(context, &memHandle);

if (result == cwNoErr)
    /* Resize factory settings handle to size of our prefs */
    result = CWResizeMemHandle(context, memHandle,
                               sizeof(SamplePref));

if (result == cwNoErr)
    /* get pointer to prefs data */
    result = CWLockMemHandle(context, memHandle, FALSE,
                           (void**)&factoryPrefs);

if (result == cwNoErr)
{
    /* initialize all settings in factoryPrefs to
       default values here, ex:

        factoryPrefs->SomeSetting = SomeValue; */

    /* unlock handle */
    CWUnlockMemHandle(context, memHandle);
    /* note: errors here intentionally ignored */
}

return (result);
}
```

---

### **reqByteSwapData Request**

When opening a project that contains preference data that was created on a platform having different byte ordering conventions, the IDE sends a [reqByteSwapData](#) request to the matching settings panel plug-in. In response, a plug-in should swap the byte ordering each element of its settings data larger than a byte. The data to be swapped resides in the current settings data handle, and should be modified in place.

In some cases, in order to swap its data correctly, a plug-in must know its current byte ordering. This happens when settings data includes a count or an offset. For example, settings data that includes an array of structures preceded by a count requires knowledge of the current byte ordering, in order to properly interpret the count. Similar considerations apply when settings data includes variable-size items preceded by a length (larger than a byte).

In such cases, a Mac OS plug-in can examine the `toEndian` field of the [PanelParameterBlock](#). Currently, the API headers do not define the values of this field, but it will contain 1 for “big-endian” (most significant byte at lowest address, like Motorola CPUs), and 2 for “little-endian” (most significant byte at highest address, like Intel CPUs).

---

**NOTE** Currently, there is no way for Windows plug-ins to determine the byte ordering in effect at the time of a [reqByteSwapData](#) request. This will be remedied in CodeWarrior Pro 6.

---

Plug-ins will not receive this request unless they set the [supportsByteSwapping](#) flag in their panel flags. Plug-ins should support this request whenever possible, to facilitate project file portability.

[Listing 8.11](#) illustrates how to respond to a [reqByteSwapData](#) request.

---

### **Listing 8.11 Handling a `reqByteSwapData` request**

---

```
CWResult ByteSwapData(CWPluginContext context)
{
    CWResult      result = cwNoErr;
    CWMemHandle  memHandle = NULL;
    SamplePref   *prefsPtr;

    /* Get a pointer to the current prefs */
    result = CWPanelGetCurrentPrefs(context, &memHandle);
    if (result == cwNoErr)
        result = CWLockMemHandle(context, memHandle,
                                FALSE, (void**)&prefsPtr);
```

---

```
// Swap the data
ByteSwapShort(&prefsPtr->ShortSetting);
ByteSwapLong(&prefsPtr->LongSetting);
}

// Relinquish our pointer to the prefs data
if (memHandle)
    CWUnlockMemHandle(context, memHandle); // error is ignored

return result;
}
```

---

### **reqValidate Request**

When the IDE sends the [reqValidate](#) request, a plug-in should examine its settings data and set three flags indicating which actions should be taken by the IDE, due to changes in a plug-in's settings data.

The three flags indicate:

- that a recompile of project files is required
- that the current target should be re-linked
- that the IDE should re-scan for files within the project paths

To determine which flags should be set, a plug-in examines its current settings data handle and compares it to the most recently saved (original) settings data.

---

**NOTE** There is a fourth “reparse” flag, which indicates that the browser information should be rebuilt by re-parsing the project source files. This flag is not currently used, however, and should be ignored by plug-ins.

---

Windows

On Windows, a panel calls [CWPanelSetRecompileFlag](#), [CWPanelSetRelinkFlag](#), and [CWPanelSetResetPathsFlag](#) to indicate whether the settings data necessitates a recompile, a relink, or re-scan for project files, respectively. The plug-in obtains its current settings handle by calling [CWPanelGetCurrentPrefs](#). It

obtains the most recently saved settings by calling  
[CWPanelGetOriginalPrefs](#).

- Mac OS On Mac OS, a panel modifies the `recompile`, `relink`, and `reset` fields of the panel parameter block directly to indicate whether the settings data necessitates a recompile, a relink, or re-scan for project files. The IDE passes the plug-in's current settings handle in the `currentPrefs` field of the [PanelParameterBlock](#), and the most recently saved settings in `originalPrefs`.

### **reqUpdatePref Request**

When opening a project, the IDE compares the version number in each settings handle stored in the project to the version number returned by its associated plug-in in the [PanelFlags Structure](#) `dataversion` field. When a settings handle contains older data than the version supported by its plug-in, the IDE sends the plug-in a [reqUpdatePref](#) request.

- 
- NOTE** Remember that a panel plug-in is required to store its settings data version number in the first 16-bit integer of its settings handle.
- 

In response, a plug-in should convert its settings data from its current version to the most up-to-date version. One way to accomplish this is to load the current preferences handle with factory defaults, and then copy any fields from the older settings handle that still apply. [Listing 8.12](#) illustrates this technique.

It is possible for the IDE to call a settings panel with data older than it understands. When this happens, a settings panel should return the result `kBadPrefVersion`.

### **Listing 8.12 Handling a reqUpdatePref request**

---

```
CWResult UpdatePref(CWPluginContext context)
{
    typedef struct SamplePref_V1 {
        short version;          // version # of pref information
        short projtype;         // project type
        char outfile[32];       // output file name
    } SamplePref_V1;
```

---

```
typedef struct SamplePref_V2 {
    short version;          // version # of pref information
    short projtype;         // project type
    char  outfile[32];      // output file name
    Boolean linksym;       // generate SYM file
} SamplePref_V2;

CWResult result = cwNoErr;
CWMemHandle memHandle = NULL;
SamplePref *currentPrefs;

result = CWPanelGetCurrentPrefs(context, &memHandle);
if (result == cwNoErr)
    result = CWResizeMemHandle(context, memHandle,
                                sizeof(SamplePref));
if (result == cwNoErr)
    result = CWLockMemHandle(context, memHandle, FALSE,
                             (void**) &currentPrefs);
if (result == cwNoErr)
{
    // All preferences must begin with a version field
    switch (currentPrefs->version)
    {
        case 1:
            // convert from version 1 to the current version
            // copy the current settings
            SamplePref_V1 v1 = * (SamplePref_V1 *) currentPrefs;
            // convert current settings to default
            MakeDefaultPrefs(currentPrefs);
            // reload old data
            currentPrefs->projtype = v1.projtype;
            strcpy(currentPrefs->outfile, v1.outfile);
            break;

        case 2:
            // convert from version 2 to the current version
            // copy the current settings
            SamplePref_V2 v2 = * (SamplePref_V2 *) currentPrefs;
            // convert current settings to default
            MakeDefaultPrefs(currentPrefs);
            // reload old data
            currentPrefs->projtype = v2.projtype;
```

```
currentPrefs->linksym = v2.linksym;
strcpy(currentPrefs->outfile, v1.outfile);
break;

// Insert other version cases here

case SAMPLEPANELDATAVERSION:
    // We are already current. Do nothing.
    break;

default:
    result = kBadPrefVersion;
    break;
}
}

if (memHandle != NULL)
    CWUnlockMemHandle(context, memHandle);
return (result);
}
```

---

#### **reqSetupDebug Request**

The IDE sends the [reqSetupDebug](#) request when the user selects **Project > Enable Debugger** or **Project > Disable Debugger**. In response to the [reqSetupDebug](#) request, the panel should determine whether debugging is enabled and make any necessary changes to debugging-related settings.

For example, the setting panel for a compiler might disable optimization, or the panel for a linker might enable output of additional symbolic information, when debugging is enabled. Panels which do not utilize this request can simply return paramErr.

**Windows** On Windows, a panel determines whether debugging is enabled by calling [CWPanleGetDebugFlag](#).

**Mac OS** On Mac OS, a panel determines whether debugging is enabled by examining the debugOn field of the panel parameter block.

---

**NOTE** The value reported by [CWPanleGetDebugFlag](#) at the time of this request reflects the action taken by the user. For example, if the

user enables debugging, the value returned at the time of the request will be true.

---

#### **reqRenameProject Request**

The IDE sends the [reqRenameProject](#) request when it creates a new project from stationery. The IDE does *not* attempt to detect and report instances of the user renaming a project, using Windows Explorer or Mac OS Finder.

Plug-ins which use the project name as the basis for other settings, such as the name of an output file or a target executable, may wish to update settings data in response to this request. Many plug-ins, however, need not respond to this request. Panels which do not utilize this request can simply return paramErr.

---

**NOTE** Mac OS plug-ins can obtain the previous name of the project by examining the name portion of the `oldtargfile` file specification in the [PanelParameterBlock](#).

---

## **Handling User Interaction**

The IDE sends only one request to both Windows and Mac OS plug-ins related to handling user interaction. However, Mac OS plug-ins receive numerous additional requests for custom dialog items, for drag and drop operations, and for handling AppleEvents. See [“Creating Settings Panel Plug-ins on Mac OS”](#) for details.

#### **reqItemHit Request**

Most of the functionality provided by the controls appearing in a settings panel is provided by the IDE automatically. The IDE, in cooperation with the host operating system, provides standard expected behavior for a panel’s UI controls. However, the IDE does not know about the meaning of individual controls, or about the relationships among them.

---

**NOTE** On Mac OS, a panel is responsible for the operation of certain user interface items. For example, Mac OS plug-ins must enforce

---

exclusivity among radio buttons, and must toggle the values of checkboxes. See [Responding to Item Hits](#) for more information.

---

The IDE sends the [reqItemHit](#) request, whenever a change has been made to a panel control by the user. In response, plug-ins can update other panel items accordingly, to indicate changes in dependent controls and internal panel state.

For example, a settings panel for a compiler might offer a checkbox to enable optimization, and a popup menu for selecting the optimization level. When the user toggles the state of the optimization checkbox, the settings panel should enable or disable the optimization level popup as appropriate.

As another example, a settings panel might present a button for selecting a folder. In response to an [reqItemHit](#) request in the button, the panel might call [CWPanlChooseRelativePath](#) to obtain a folder specification, and store the result in its plug-in data. After the user selects a folder, the panel might convert it to a human-readable string using [CWPanlGetRelativePathString](#), and store it in a static text item, for user feedback.

Every time the IDE calls a settings panel to handle a [reqItemHit](#) request, the plug-in should be sure to set the revert and factory flags. The revert flag should be set true if the settings, as currently configured, differ from the most recently committed changes, as stored in the original settings handle. The factory flag should be set if the current settings differ from the factory settings.

- |         |   |
|---------|---|
| Windows | On Windows, plug-ins can set the revert and factory flags by calling <a href="#">CWPanlSetRevertFlag</a> and <a href="#">CWPanlSetFactoryFlag</a> .   |
| Mac OS  | On Mac OS, to return the factory and revert flags to the IDE, a plug-in modifies the <code>canRevert</code> and <code>canFactory</code> fields of the <a href="#">PanelParameterBlock</a> directly. |

## Importing and Exporting Settings Data

Beginning with CodeWarrior Pro, release 5, the IDE provides an option to export projects and their attendant settings as XML. When the user selects **File > Import Project** or **File > Export Project**, the

IDE reads or writes the contents of a project file as XML, and then requests that each settings panel enabled by the current project import or export its settings data.

It is beyond the scope of this document to describe XML in detail. In general, however, to import or export its settings as XML, a panel plug-in must map its internal setting data onto standard data types, and associate a unique name with each setting.

**WARNING!**

Because settings names must be unique within a given XML file (shared by all project plug-ins), it is important for plug-ins to choose distinctive names for their settings. A good rule of thumb is to use a setting name prefix that reflects the name of the tool vendor and the tool. For example, all setting names for data exported by the Acme PROLOG compiler's settings panel might start with the prefix "AcmeProlog..."

### XML Setting Types

Settings data may be composed of any of the following types which may be read and written using the indicated routines:

<b>Setting type:</b>	<b>Read and write using:</b>
boolean	<a href="#">CWReadBooleanSetting</a> <a href="#">CWWriteBooleanSetting</a>
integer (long integer)	<a href="#">CWReadIntegerSetting</a> <a href="#">CWWriteIntegerSetting</a>
floating point (double)	<a href="#">CWReadFloatingPointSetting</a> <a href="#">CWWriteFloatingPointSetting</a>
string	<a href="#">CWReadStringSetting</a> <a href="#">CWWriteStringSetting</a>
relative path	<a href="#">CWReadRelativePathSetting</a> <a href="#">CWWriteRelativePathSetting</a>

The routines listed above read and write *top-level* settings. Top-level settings appear at the "root" level of a plug-in's XML data. XML supports nested complex data types, such as arrays and structures.

Top-level settings are contained directly in a panel's XML stream, never in nested arrays and structures.

In some cases, a plug-in may have to use a somewhat different data type to import or export its settings than it uses internally. For example, enumerations can be handled using either integer types or strings.

### XML Structured Types

In addition to the simple types listed above, a plug-in may group its settings data into arrays and structures. The process of reading or writing such a structured type involves additional steps beyond those necessary to read and write simple types.

To read or write a structured type, a plug-in must first obtain the ID of a collection object, whether structure or array. Collection objects contain other XML elements grouped into arrays and structures. A plug-in obtains an ID for a top-level ("root") array or structure by calling [CWGetNamedSetting](#).

The ID returned by [CWGetNamedSetting](#) identifies a container created by the IDE (when writing XML), or present in the XML input stream (when reading XML). The plug-in uses this ID to refer to the container, when adding items to it or when extracting values from it.

[CWGetNamedSetting](#) operates differently when the plug-in is responding to a [reqReadSettings](#) request than it does when responding to a [reqWriteSettings](#) request. When reading XML, [CWGetNamedSetting](#) will return an error when the plug-in asks for a setting that is not present in the XML input stream. When writing XML, [CWGetNamedSetting](#) will create the named setting, and return an ID for it.

- 
- NOTE** Although [CWGetNamedSetting](#) is normally used for structured data items, it can be used to read and write single name-value pairs. In fact, the calls for reading and writing simple values (beginning with the `CWWrite...` and `CWRead...` prefixes) are implemented using [CWGetNamedSetting](#), followed by calls to the appropriate

CWGet... or CWSet routine. However, it is simpler and easier to just use the CWWrite and CWRead routines.

---

To add items to a container (when writing settings data), a plug-in calls one of two routines, depending upon container type. For structures, the plug-in calls [CWGetStructureSettingField](#). For arrays, the plug-in calls [CWGetArraySettingElement](#). These calls create a new structure field or array element, respectively, and return its ID.

---

**NOTE** Currently, the IDE requires that array elements be created in sequence, starting with an index of 0, working toward higher indexes.

---

Both routines can be used to read XML data as well, in response to a [reqReadSettings](#) request. [CWGetStructureSettingField](#) and [CWGetArraySettingElement](#) will return an error if the named setting is not found in the parent XML container, instead of creating a new structure field or array element.

### XML Structured Type Elements

IDs returned by either routine can then be used with the following calls to read and write an individual field or array element that consists of a single value:

---

<b>Setting type</b>	<b>Read and write using:</b>
boolean	<a href="#">CWGetBooleanValue</a> <a href="#">CWSetBooleanValue</a>
integer (long integer)	<a href="#">CWGetIntegerValue</a> <a href="#">CWSetIntegerValue</a>
floating point (double)	<a href="#">CWGetFloatingPointValue</a> <a href="#">CWSetFloatingPointValue</a>
string	<a href="#">CWGetStringValue</a> <a href="#">CWSetStringValue</a>
relative path	<a href="#">CWGetRelativePathValue</a> <a href="#">CWSetRelativePathValue</a>

---

In addition to the preceding calls which read and write simple types, a plug-in may call [CWGetStructureSettingField](#) and [CWGetArraySettingElement](#) again to read and write additional nested structures and array elements.

#### **reqWriteSettings Request**

In response to a [reqWriteSettings](#) request, a plug-in should write its settings data as XML. plug-ins write individual settings using the appropriate CWWrite... calls, and structures and arrays using [CWGetNamedSetting](#), [CWGetStructureSettingField](#) and [CWGetArraySettingElement](#).

See the SDK sample projects for an example of how to write settings data to XML.

#### **reqReadSettings Request**

In response to a [reqReadSettings](#) request, a plug-in should read its settings data from XML into its settings handle. When reading its settings, a plug-in normally starts by initializing its current settings handle with factory default values. The plug-in then reads items from the XML input stream, skipping fields which can not be found. This process ensures that the new settings data contains the correct (current) version number, as well as default values for all fields, which may not be present in the XML when reading older settings data.

Plug-ins read individual settings using the appropriate CWRead... calls, and structures and arrays using [CWGetNamedSetting](#), [CWGetStructureSettingField](#) and [CWGetArraySettingElement](#).

Plug-ins that support reading and writing settings data as XML must set the [supportsTextSettings](#) flag in their [Settings Panel Flags](#) in order to receive the [reqWriteSettings](#) and [reqReadSettings](#) requests. Otherwise, the IDE will not issue these requests.

See the SDK sample projects for an example of how to read settings data from XML.

# Managing Panel Items

When responding to user interaction, a panel often changes the state of its controls. This section describes routines provided for getting and setting control values, for enabling and disabling items, for validating user input, and for drawing and highlighting items.

This section covers the following topics:

- [Getting and Setting Control Values](#)
- [Enabling and Disabling Items](#)
- [Validating Input](#)
- [Managing Input Focus](#)
- [Redrawing Panel Items](#)

## Getting and Setting Control Values

The IDE provides routines for getting and setting the values of panel controls. A handful of routines suffice for manipulating a wide range of control types.

### Getting and setting numeric controls

Plug-ins obtain the values of numeric controls, such as checkboxes and radio buttons using [CWPanelGetValue](#). The range of values returned will vary according to the control type. For example, checkboxes return only the value 0 or 1, depending upon whether they are cleared or checked, respectively.

Plug-ins change the values of numeric controls using [CWPanelSetItemValue](#). Valid values to use depend upon the control type, as above.

### Getting and setting text items

To set the text in an editable text field, static text field, or title of a dialog item, use [CWPanelSetText](#). To get the text of an item, use [CWPanelGetItemText](#). To get and set the text of an item as a handle instead, call [CWPanelGetItemTextHandle](#) and [CWPanelSetItemTextHandle](#), respectively.

## **Creating Settings Panel Plug-ins**

### *Enabling and Disabling Items*

---

**NOTE** This is especially useful on Mac OS, where the strings returned by [CWPanelGetItemText](#) and [CWPanelSetItemText](#) are limited to 255 characters.

---

Plug-ins use [CWPanelGetValue](#) to get the value of a text field as an integer value. [CWPanelGetValue](#) will return an error if the text of an item is not a valid number. Plug-ins set a text field to a numeric value using [CWPanelSetValue](#).

#### **Limiting text input**

Plug-ins can constrain the maximum text allowed in a text field using [CWPanelSetItemMaxLength](#). For example, a text entry field for a DOS file name might have a maximum length of 8 characters. Plug-ins can determine the maximum text entry length for a field using [CWPanelGetItemMaxLength](#).

## **Enabling and Disabling Items**

To enable and disable items, use [CWPanelEnableItem](#). A disabled item is visible, but greyed, and its value can't be changed by the user. An enabled item is drawn normally and accepts input from the user. Plug-ins commonly enable and disable items in response to [reqItemHit](#) requests.

For example, a preference panel for a compiler might provide a checkbox for enabling and disabling optimization for a particular CPU type. When checked, an additional popup menu specifying the model of CPU to optimize for would be enabled. When unchecked, the popup would be disabled. Enabling and disabling the popup would normally happen in response to [reqItemHit](#) requests.

To hide or show an item, use [CWPanelShowItem](#). When possible, items should be enabled and disabled, not hidden. It is normally only appropriate to hide panel items when an item will not be enabled at any time during the use of its panel, or when multiple sets of controls are switched into a small area, as with tab controls.

## Validating Input

The IDE provides several routines that can be used to validate user input, and to correct it if necessary.

To obtain the content of a text item, call [CWPanellGetItemText](#). Plug-ins typically call [CWPanellGetItemText](#) in response to an [reqItemHit](#) event, to determine whether the text entered is valid. The returned text can be analyzed, and if invalid, the plug-in can inform the user. Alternatively, the plug-in can discard an invalid entry by replacing entered text with its previous value using [CWPanellSetItemText](#).

The IDE also offers support for validating numeric entry. To ensure that a text field contains a valid decimal number, use [CWPanellGetItemValue](#). When applied to a text item, this call attempts to convert the text of the item to a number and return its value. If the conversion fails, it returns an error. If the value returned is out of range, a plug-in can install a valid value in a field using [CWPanellSetItemValue](#).

In some cases, it may also be useful to limit the length of entered text. Plug-ins call [CWPanellGetItemMaxLength](#) to determine the maximum allowed text entry length for a text field, and [CWPanellSetItemMaxLength](#) to change it. To determine the actual length of the text in a dialog item, call [CWPanellGetItemText](#).

## Managing Input Focus

Use [CWPanellActivateItem](#) to change the input focus based on a mouse, keyboard, or other event. [CWPanellActivateItem](#) ensures that the specified item receives subsequent keystrokes.

Plug-ins may find it necessary to track which item has input focus, especially in the case of custom items. Plug-ins must know which item has input focus when redrawing custom items, in order to draw input focus highlighting correctly.

- Mac OS      Previously, 'DITL'-based Mac OS plug-ins were required to highlight custom items as they acquired and lost input focus. See ["Managing Input Focus \(Mac OS\)" on page 393](#). 'DITL'-based panels are deprecated, and custom items are not currently supported.

## Redrawing Panel Items

Panels can request that an item be redrawn by “invalidating” the area occupied by the item on screen. Panels can also inform the IDE and operating system that part of the screen no longer needs to be redrawn by “validating” it.

To invalidate the drawing region associated with a dialog item, use [CWPanelInvalItem](#). To validate an item’s drawing region use [CWPanelValidItem](#). Both routines require a rectangular specification for the area to invalidate, in local coordinates. Mac OS plug-ins can call [CWPanelGetItemRect](#) to get an item’s bounding box.

Most plug-ins do not need to call [CWPanelInvalItem](#) or [CWPanelValidItem](#). The IDE takes care of drawing and redrawing standard controls. [CWPanelInvalItem](#) and [CWPanelValidItem](#) are most useful with custom controls which provide secondary representations of settings data.

For example, a custom item that displays a preview of a setting specified in a popup menu might need to be redrawn when the associated popup menu setting changes. In response to an [reqItemHit](#) event in the popup menu, a plug-in can invalidate the custom preview item.

This has the side effect of putting all drawing logic in the drawing routine for the custom item, rather than distributing it throughout the panel code. This is usually desirable from a code organization and maintenance standpoint. It also avoids subtle redraw problems, since all custom item drawing uses identical code.

# Creating Settings Panel Plug-ins on Windows

---

This chapter discusses how to create settings panel plug-ins for Windows-hosted versions of the CodeWarrior IDE.

## Creating Settings Panel Plug-ins on Windows Overview

This chapter covers settings panel plug-in development for Windows hosted versions of the IDE. There are relatively few Windows-specific development issues.

For information about settings panels in general, see [“Creating Settings Panel Plug-ins” on page 331](#). For information about Mac OS settings panels, see [“Creating Settings Panel Plug-ins on Mac OS” on page 383](#).

This chapter covers the following topics:

- [Help Information Entry Point](#)
- [Handling Panel Requests from the IDE](#)
- [Panel Resource Construction](#)

## Help Information Entry Point

Windows panel plug-ins may optionally implement a help information entry point. This entry point returns the name of a WinHelp help file to be opened by the IDE when the user requests help for a panel.

The help information entry point has the following declaration:

**Listing 9.1 CWPlugin\_GetHelpInfo entry point declaration**

---

```
CWPLUGIN_ENTRY (CWPlugin_GetHelpInfo)(const CWHelpInfo**  
helpInfo);
```

---

The entry point returns a pointer to a `CWHelpInfo` structure to the IDE. The structure returned must remain valid for the entire time the plug-in is loaded. Plug-ins should return a pointer to a statically allocated structure, containing the name of the help file, and the current `CWHelpInfo` structure version.

[Listing 9.2](#) illustrates the process.

**Listing 9.2 Example CWPlugin\_GetHelpInfo entry point**

---

```
CWPLUGIN_ENTRY (CWPlugin_GetHelpInfo)(const CWHelpInfo**  
helpInfo)  
{  
    static CWHelpInfo MyHelpInfo =  
    {  
        kCurrentCWHelpInfoVersion,  
        "MyHelp.hlp"  
    };  
  
    *helpInfo = &MyHelpInfo;  
    return 0;  
}
```

---

## Handling Panel Requests from the IDE

This section discusses the following topics:

- [Handling Panel Requests](#)
- [Creating a Panel's Interface](#)

### Handling Panel Requests

When the IDE calls a panel plug-in, it passes the plug-in a context parameter of type [`CWPluginContext`](#). The IDE and plug-in use

this data, described in “[Main Entry Point context Parameter](#)” on [page 77](#), to pass information between them.

Like other plug-in types, Windows settings panel plug-ins should call [CWGetPluginRequest](#) to determine what the IDE is asking the plug-in to do. The plug-in should then handle the request, typically using code similar to that presented in [Listing 8.4](#).

The example in [Listing 8.4](#) handles the important requests that most Windows settings panels should handle. These include:

- reqInitialize
- reqTerminate
- reqInitDialog
- reqTermDialog
- reqPutData
- reqGetData
- reqItemHit
- reqValidate
- reqGetFactory
- reqUpdatePref

The following requests are optional, but plug-ins should handle them correctly if at all possible:

- reqByteSwapData
- reqReadSettings
- reqWriteSettings

---

**NOTE** A plug-in must set the [supportsByteSwapping](#) and [supportsTextSettings](#) flags in its panel flags to receive the requests listed above.

---

Additional requests that some plug-ins may want to handle include:

- reqFirstLoad
- reqRenameProject
- reqSetupDebug

- 
- TIP** The other requests listed in the enumeration in DropInPanelWin32.h are Mac-only, and will not be sent to Windows-hosted plug-ins.
- 

## Creating a Panel's Interface

When the panel receives a [reqInitiDialog Request](#), it should call [CWPanelAppendItems](#) specifying the ID number of a dialog resource describing the panel's user interface items. In the example Windows settings panel project, this ID is 1000, but can be any number. The IDE loads the specified dialog resource and uses it to construct the panel user interface. Dialog resources are normally linked into the final compiled plug-in DLL.

For more information on the dialog panel resource creation, refer to [“Panel Resource Construction” on page 378](#).

# Panel Resource Construction

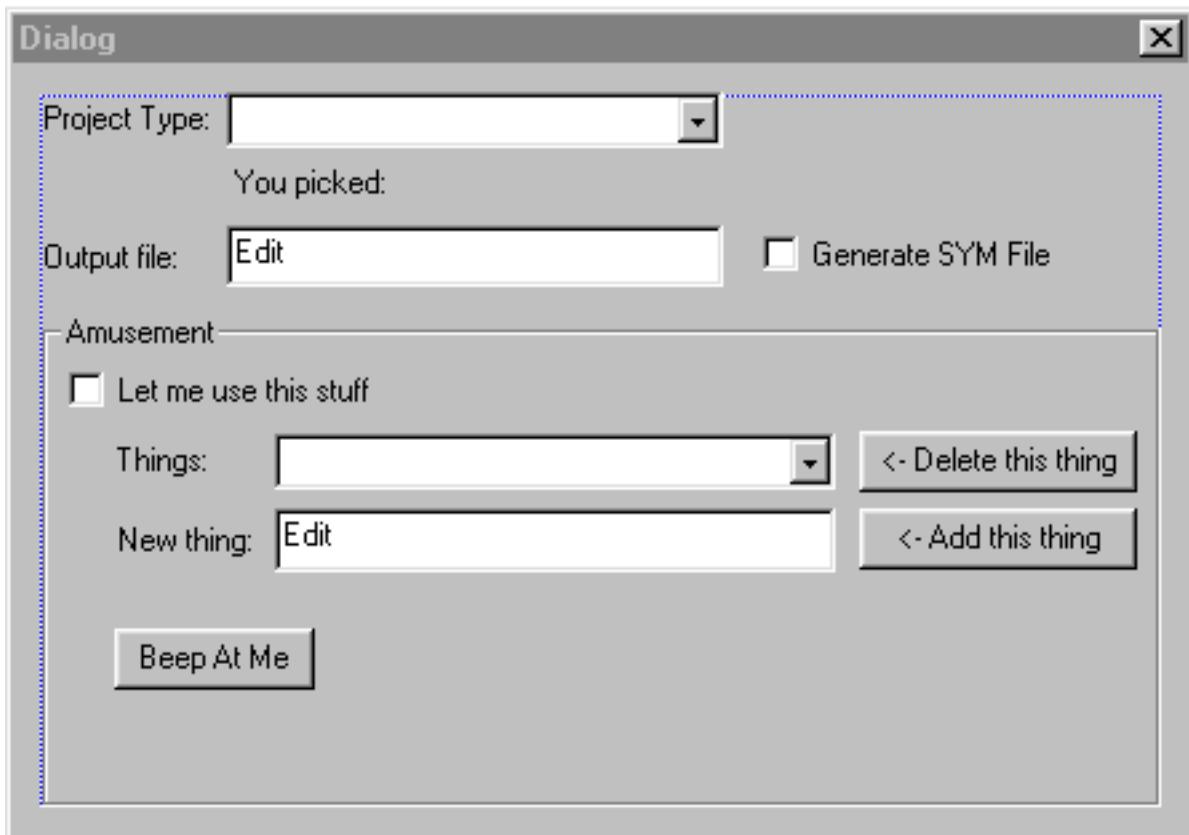
The IDE constructs a panel's user interface from a Windows dialog resource, which should be linked into your plug-in's DLL. The items specified in a panel's dialog resource are added to the IDE's **Target Settings** dialog when the panel is displayed.

## Dialog Resource Construction

You create a panel's user interface using standard Windows dialog template resources. These can be created using any visual resource editor. They are normally compiled and linked into a plug-in DLL binary by including the “.rc” file produced by a resource editor in the plug-in project.

The sample panel from the Windows panel example code shown in [Figure 9.1](#) was created using Microsoft Visual C++ 5.0.

**Figure 9.1 Sample Panel Dialog**

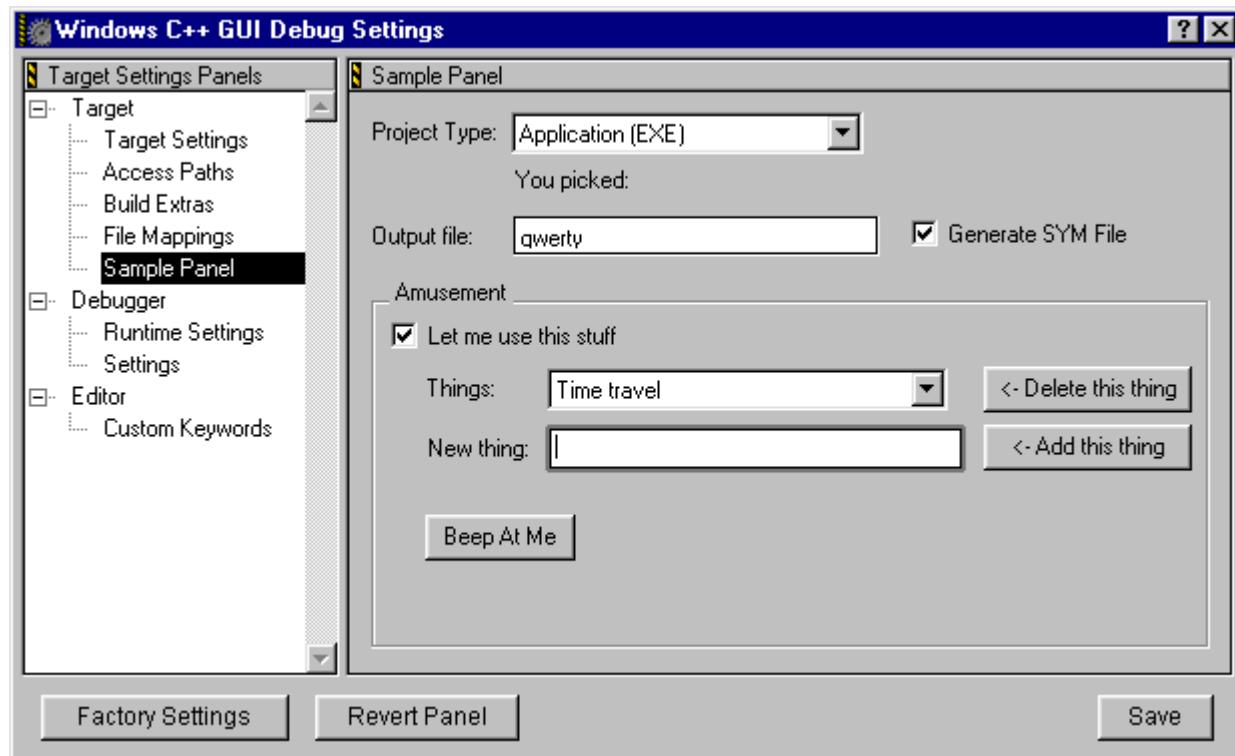


[Figure 9.2](#) illustrates what the panel looks like when running in the IDE.

## Creating Settings Panel Plug-ins on Windows

### Control ID Assignment

**Figure 9.2** Finished Preferences Panel



## Control ID Assignment

The IDE identifies and tracks controls in your panel by ID number. ID numbers are required for programmatic access to a panel's controls. For every item in your panel that you want to manipulate using API routines, you must assign a unique ID number, as shown in [Figure 9.3](#).

**Figure 9.3 Unique number assignment**



## Panel Layout Requirements

In order for a panel to fit in the **Target Settings** dialog box correctly, it should be defined with the following characteristics:

- **Font:** MS Sans Serif
- **Font Size:** 8
- **Horizontal Size:** 292 dialog units
- **Vertical Size:** 178 dialog units

## Supported Control Types

Some but not all of the native Windows dialog controls are supported in panel interfaces. The following controls are supported:

- Push button
- Check box
- Radio button
- Group box
- Edit text
- Static text
- Combo box

Custom user items and list boxes are not yet supported, but may be in a future release.

## **Supported Control Styles**

Many Windows control styles are not currently supported. The following control styles are supported:

### **Button styles**

- Push buttons (BS\_PUSHBUTTON, BS\_DEFPUSHBUTTON)
- Radio buttons (BS\_RADIOBUTTON, BS\_AUTORADIO)
- Check boxes (BS\_CHECKBOX, BS\_AUTOCHECKBOX)
- Group boxes (BS\_GROUPBOX)
- Visible (WS\_VISIBLE)
- Disabled (WS\_DISABLED)

### **Edit Text styles**

- Left justified (ES\_LEFT)
- Center justified (ES\_CENTER)
- Right justified (ES\_RIGHT)
- Multiline (ES\_MULTILINE) but only when ES\_AUTOHSCROLL is not specified
- Autoscrolling (ES\_AUTOVSCROLL or ES\_AUTOHSCROLL)
- Visible (WS\_VISIBLE)
- Disabled (WS\_DISABLED)

### **Static Text styles**

- Left justified (SS\_LEFT)
- Center justified (SS\_CENTER)
- Right justified (SS\_RIGHT)
- Visible (WS\_VISIBLE)
- Disabled (WS\_DISABLED)

### **Combo Box styles**

No specific styles are supported. All combo boxes are created as Drop-down list boxes (CBS\_DROPDOWNLIST).

- Visible (WS\_VISIBLE)
- Disabled (WS\_DISABLED)

# Creating Settings Panel Plug-ins on Mac OS

---

This chapter discusses how to create settings panel plug-ins for Mac OS-hosted versions of the CodeWarrior IDE.

## Creating Settings Panel Plug-ins on Mac OS Overview

This chapter covers settings panel plug-in development for Mac OS hosted versions of the IDE. The Mac OS version of the settings panel API is more complex than the Windows version. Most of the extra complexity pertains to event handling and custom panel items.

Other general differences include plug-in state, which is opaque on Windows, but which requires direct access by plug-ins on Mac OS. Mac OS plug-ins also use standard Mac OS toolbox Handles, rather than CodeWarrior [CWMemHandles](#). Specific functionality differences include support for AppleScript and drag and drop UI events on Mac OS.

---

**NOTE** Future versions of CodeWarrior will reduce the differences between the Mac OS and Windows versions of the API.

---

For information about settings panels in general, see [“Creating Settings Panel Plug-ins” on page 331](#). For information about Windows settings panels, see [“Creating Settings Panel Plug-ins on Windows” on page 375](#).

This chapter covers the following topics:

- [Settings Panel Resources](#)

- [Requests from the IDE](#)

---

**NOTE** The cross-references in this chapter refer to the Windows versions of API routines. This is because the bulk of the reference content appears under the Windows entries, to avoid duplication. Remember that the actual routine name for most of the Mac OS routines starts with 'Panl', not 'Panel'.

---

## Settings Panel Resources

This section describes platform-specific resources required by Mac OS settings panel plug-ins.

---

**NOTE** The CW plug-in API package includes resource templates for special panel resources in Resorcerer™ and Rez formats.

---

This section covers these topics:

- [User Interface Resources](#)
- [Providing Balloon Help](#)
- [Apple Event Dictionary \('aete'\) Resource](#)

## User Interface Resources

In previous versions of the IDE and plug-in API, Mac OS panel user interfaces were constructed from 'DITL' resources. This method of constructing a panel's user interface is now officially deprecated. New Mac OS panel plug-ins should use 'PPop' resources.

Instead of passing the resource number of a 'DITL' resource when calling [CWPanelAppendItems](#), plug-ins pass the number of a 'PPop' resource. Use Metrowerks Constructor (the visual editor for user interfaces built upon the PowerPlant class library) to design settings panel user interfaces. Note that 'PPop' support is minimally implemented and still evolving.

As a consequence of switching from 'DITL' resources to 'PPop' resources, some portions of the API which pertain to the operation of custom controls have become temporarily obsolete. Specifically,

requests for drawing, activating, and deactivating items, as well as managing the edit menu for custom items are obsolete, since custom items are not currently supported using 'PPob' resources. For completeness, these requests are still documented.

---

**NOTE** The IDE still supports 'DITL' resources for those plug-ins that absolutely require these features. Plug-ins implemented using 'DITL' resources may, however, experience cosmetic UI problems and may not be supported in future IDEs.

---

### **Panel Dimensions**

All of a panel's controls should be nested in an `LView` layout, with the following dimensions:

- width: 440 pixels
- height: 290 pixels

### **Supported PowerPlant Controls**

The IDE supports the following types of PowerPlant controls and attachments in settings panel user interfaces:

---

<b>User interface function:</b>	<b>Constructor class:</b>
General drawing area	<code>LView</code>
Descriptive static text	<code>LCaption</code>
Editable text entry field	<code>LEditingText</code> <b>Important:</b> all edit fields used in panels must have a class ID of ' <code>PEDT</code> '. Specify this using the <b>Property Inspector</b> in Constructor.
Simple action button	<code>LPushButton</code>
Two-valued check box	<code>LCheckbox</code>
One-of-many selector	<code>LRadioButton</code>

User interface function:	Constructor class:
Group view for radio buttons (enforces mutual exclusion among values of nested controls)	LRadioGroupView
Iconic action buttons and popup menus	LBevelButton
Popup menus	LPopupButton
Visual separator	LSeparatorLine
Labelled control group	LTextGroupBox
Checkbox-enabled control group	LCheckBoxGroupBox
Popup-enabled control group	LPopupGroupBox
Background drawing attachment	LAttachment
Balloon help content	ContextHelpAttachment

The IDE handles most of these controls automatically. Panel user interface elements may require initialization, which can be done when responding to [reqInitDialog Requests](#).

### Setting Panel Background

Panels must attach a background drawing attachment to their top-level (outermost enclosing) LView, in order to display the proper gray colored background. To do this, create an LAttachment with a type code of 'PBkg' and set it to respond to the msg\_DrawOrPrint message. Use the **Layout > Show Object Hierarchy** command in Constructor to ensure that the attachment is bound to the view, not a subview.

### Important Restrictions

Note the following important restrictions on using PowerPlant controls:

- Metrowerks will not support the use of any controls other than those listed above (regardless of what you may discover by reverse engineering shipping panels).

- The IDE's support for particular controls is still changing, and is a likely area for future revisions. If you take advantage of undocumented features, your code will break in the future.
- Custom controls previously implemented using 'DITL' user items are not currently supported. Few panels require custom controls.

## Providing Balloon Help

Panels can provide Balloon Help by adding a `ContextHelpAttachment` to each item in a panel. To do this, ensure that the `ContextHelpAttachment` 'CTYP' provided with the plug-in SDK is installed in the 'Custom Types' folder within the Constructor application folder.

Once installed, an attachment of type `ContextHelpAttachment` should appear on the **Attachments** pane in Constructor catalog window (accessible via **Window > Catalog**). To add Balloon Help to a panel, drag a `ContextHelpAttachment` to a panel item. To ensure association between the proper `ContextHelpAttachment` attachments and panel items, drag items to the Hierarchy window (**Layout > Show Object Hierarchy**) rather than the usual view editor window. Then, use the **Property Inspector** window to edit the help text for each panel item.

Each `ContextHelpAttachment` supplies three text strings displayed when Balloon Help is turned on: one for when the item is enabled, one for when the item is disabled, and one for when the item is checked. Not all strings make sense for all controls. For example, only the enabled string is used for edit text items.

## Apple Event Dictionary ('aete') Resource

Mac OS settings panels support manipulation of their settings data by providing an '`aete`' AppleScript terminology (dictionary) resource. In the general case, an '`aete`' resource specifies many classes and events handled by an application, and the user-written AppleScript syntax for them.

In the specific case of a CodeWarrior settings panel plug-in, the '`aete`' resource is much simpler. Each plug-in declares one class, and no events. The class contains one property for each component

of a settings panel's data. The class name should reflect the general purpose of the settings panel and its data.

'aete' resources may be constructed using any of the following tools:

- Resorcerer, using its built-in 'aete' editor
- ResEdit, using a freely-available 'aete' plug-in (see Apple's developer CDs)
- The Hypercard-based Aete Editor Stack (also on Apple's developer CDs)
- Other third party solutions, such as EightyRez (now free)
- Conventional textual source files, compiled into binary resources by tools such as Metrowerks Rez (integrated resource compiler) or Apple's MPW Rez tool

To maintain consistency, panel plug-ins should, whenever possible, match existing numeric Apple Event codes and terms used by the IDE's existing panels. Refer to the *CodeWarrior IDE User Guide* for more information on IDE AppleScripting and Apple Events.

For more information about creating 'aete' resources, see *Inside Macintosh: Interapplication Communication*. For information about standard Apple Event data types, see the *Apple Event Registry* (available on Apple's developer CDs). For information about responding to Apple Event requests, see "[Handling Apple Events](#)" [on page 396](#).

## Requests from the IDE

This section discusses IDE requests specific to Mac OS settings panels. The topics in this section are:

- [Getting a Panel Request](#)
- [Returning Panel Results to the IDE](#)
- [Handling Keyboard and Mouse Hits](#)
- [Filtering Low-Level Events](#)
- [Drawing Custom Items](#)
- [Handling Edit Menu Commands](#)

- [Handling Drag And Drop](#)
- [Handling Apple Events](#)

## Getting a Panel Request

When the IDE calls a panel plug-in, it passes the panel a [PanelParameterBlock](#). The IDE and plug-in use the parameter block to communicate information about plug-in requests.

Among the items in the parameter block is a request value, passed in the `request` field. This value specifies the action to be taken by a plug-in. Other items in the parameter block contain information relevant to specific requests, such as the current project's file specification; the current, original, and factory settings data; various flags the plug-in sets to indicate properties of its plug-in data; and so on.

[Listing 8.4](#) shows an example main entry point for both Mac OS and Windows plug-ins. The request dispatch code for Mac OS uses the `request` field of the parameter block to determine the action requested by the IDE.

## Returning Panel Results to the IDE

After servicing a request, a plug-in returns an error code to the IDE. Mac OS panel plug-ins report errors to the IDE somewhat differently than other plug-ins.

Instead of calling [CWDonePluginRequest](#), a Mac OS plug-in simply returns an error code it wishes to report to the IDE as its `main( )` function result. A plug-in reports that it has successfully acted on a request by returning `cwNoErr`. If a plug-in returns any other value, the IDE acts accordingly and reports an error to the user, if necessary.

See [Listing 8.4](#) for an example of returning results to the IDE.

## Handling Keyboard and Mouse Hits

The IDE sends the [reqHandleKey](#) and [reqHandleClick](#) requests when a custom dialog item receives a key stroke or mouse click,

respectively. [Listing 10.1](#) shows an example of how to handle keyboard hits in a custom item.

The code illustrates how to examine the Mac OS event record to determine which arrow key was pressed, modify the plug-in's settings data, and return a simulated dialog item hit to the IDE. Note that the plug-in modifies the `itemHit` field of the parameter block prior to return.

### **Listing 10.1 Handling a keyboard hit**

---

```
static void HandleKey(PanelParameterBlock *pb, EventRecord
*event)
{
    Point tempPos = sDotPosition;
    short simulatedItem = kPictItem;
    Rect itemRect, dotRect;

    CWPanlGetItemRect(pb, kPictItem, &itemRect);
    dotRect = itemRect;
    InsetRect(&itemRect, 4, 4);
    OffsetRect(&itemRect, -itemRect.left, -itemRect.top);

    switch (event->message & charCodeMask)
    {
        case 0x1C: /* Left arrow */
            if (tempPos.h-- <= itemRect.left)
                return;
            break;
        case 0x1D: /* Right arrow */
            if (tempPos.h++ > itemRect.right)
                return;
            break;
        case 0x1E: /* up arrow */
            if (tempPos.v-- <= itemRect.top)
                return;
            break;
        case 0x1F: /* Down arrow */
            if (tempPos.v++ > itemRect.bottom)
                return;
            break;
    default:
```

```
    return;
}
sDotPosition = tempPos;
InsetRect(&dotRect, 3, 3);
DrawDot(sDotPosition, &dotRect);
pb->itemHit = simulatedItem + pb->baseItems;
}
```

---

## Filtering Low-Level Events

The IDE sends the [reqFilter](#) request to allow the panel to handle raw Mac OS events. The IDE passes the plug-in a copy of the event received by its event loop, prior to handling the event. The event is passed in the event field of the panel parameter block.

Plug-ins can inspect the event record, and take any necessary action, such as modifying the state or appearance of an affected control.

Plug-ins may also modify the event record if necessary, prior to returning to the IDE. For example, if a panel completely handles an event, it should change the type of the event (the what field of the Mac OS toolbox EventRecord) to nullEvent.

Panel plug-ins typically only perform this sort of low-level event handling when implementing custom controls. Most panels do not need to respond to [reqFilter](#) requests. When possible, plug-ins should handle events in custom controls using the [reqHandleKey](#) and [reqHandleClick](#) requests. See “[Handling Keyboard and Mouse Hits](#)” on page 389.

## Responding to Item Hits

For certain standard control types, such as check boxes and radio buttons, the Mac OS hosted IDE defers implementation of some of the standard control behavior to plug-ins. This typically occurs in response to a [reqItemHit](#) request.

In the case of checkboxes, a plug-in obtains the value of a checkbox using [CWPanleGetItemValue](#), complements its value using, for example, the C logical “not” operator (' ! '), and then sets the checkbox to its new value using [CWPanleSetItemValue](#).

For radio buttons, a plug-in can set the value of each button in a group to 0 using [CWPanelSetItemValue](#), and then set the hit button to 1, again using [CWPanelSetItemValue](#). Alternatively, a plug-in can keep track of the currently selected radio button control and set it to 0, instead of setting all controls within the group to 0.

Panels can also group radio buttons using an `LRadioGroupView` control. To determine the selected radio button in the group, call [CWPanelGetItemValue](#). To set the selected radio button, use [CWPanelSetItemValue](#). This is the simplest way to manage radio buttons, but does not work for 'DITL'-based panels.

## Drawing Custom Items

This section explains how to draw custom dialog items. Custom panel correspond to “user items” in panels contructed using Mac OS dialog manager ‘DITL’ resources, but are not yet available in panels constructed from ‘PPop’ resources.

Panels which implement custom items are responsible for drawing them. The IDE asks a plug-in to draw a custom control by sending the [reqDrawCustomItem](#) request.

Plug-ins can frame custom items using [CWPanelDrawPanelBox](#), which optionally also draws a textual label for the item. Plug-ins can also use QuickDraw routines to draw custom items.

Before using any QuickDraw routines, plug-ins should first change the drawing port to the dialog window. Plug-ins can obtain the dialog window `GrafPtr` using [CWPanelGetMacPort](#).

**WARNING!**

Plug-ins should be careful to restore all port properties, such as text styles, foreground and background colors, and clipping regions, before returning to the IDE.

Plug-ins determine the rectangle of a dialog item in local coordinates by calling [CWPanelGetItemRect](#). The rectangle returned specifies the bounding rectangle of an item, which may be useful when drawing, invalidating, or hit-testing an item.

When drawing a custom item, a plug-in should also highlight the item if it is active (has input focus). Plug-ins can track the active item when responding to [reqActivateItem](#) requests.

## Managing Input Focus (Mac OS)

This section explains how to activate and deactivate custom panel items. Custom items correspond to “user items” in panels constructed using Mac OS dialog manager ‘DITL’ resources, but are not yet available in panels constructed from ‘PPop’ resources.

The IDE sends the [reqActivateItem](#) and [reqDeactivateItem](#) requests when activating and deactivating custom items, respectively. In response, a panel should highlight the specified item.

### When a custom item acquires input focus

When a custom dialog item becomes the input focus, the IDE sends the [reqActivateItem](#) request. If applicable, the panel should respond by drawing a focus highlight for the item with input focus. Otherwise, the panel can ignore this request.

### When a custom item loses input focus

When a custom dialog item loses the input focus, the IDE sends the [reqDeactivateItem](#) request. If applicable, the panel should respond by removing focus highlight from the item losing input focus. Otherwise, the panel can ignore this request.

### Tracking the active item

When responding to [reqDrawCustomItem](#), [reqFindStatus](#), and [reqObeyCommand](#) requests, plug-ins need to know which custom item is active. The IDE does not pass this information to plug-ins. Because of this, plug-ins should save the active item number in a global variable when responding to the [reqActivateItem](#) request.

## Handling Edit Menu Commands

This section explains how to manage the **Edit** menu for custom panel items. Custom items correspond to “user items” in panels

constructed using Mac OS dialog manager 'DITL' resources, but are not yet available in panels constructed from 'PPop' resources.

The IDE sends status requests, to determine which commands should be enabled when a custom control is active. The IDE also sends command requests, when a custom item is active and the user selects an enabled **Edit** menu command.

#### Determining Edit Menu Command Status

The IDE automatically handles **Edit** menu commands (**Cut**, **Copy**, **Paste**, **Clear**, **Select All**) for standard editable text fields. The IDE also supports these commands for custom controls.

To find out which **Edit** menu commands a custom dialog item handles, the IDE sends [reqFindStatus](#) requests for each custom dialog item in a panel. The IDE enables **Edit** menu items based upon the plug-in's responses.

Plug-in panels respond to the [reqFindStatus](#) request by examining the `itemHit` field of the panel parameter block. The IDE uses this field to specify an **Edit** menu command using one of these constants: [menu\\_Cut](#), [menu\\_Copy](#), [menu\\_Paste](#), [menu\\_Clear](#), and [menu\\_SelectAll](#).

Plug-ins return a 0 or 1 in the `itemHit` field (overwriting the edit command sent to the plug-in) depending upon whether it supports the specified command for the currently selected custom control. A value of 1 indicates that a plug-in supports the specified command for the current item.

To respond to [reqFindStatus](#) requests correctly, plug-ins must track which custom item has input focus. Plug-ins track the active item when responding to [reqActivateItem](#) requests. See [Managing Input Focus \(Mac OS\)](#).

#### Responding to Edit Menu Commands

To allow the panel to handle **Edit** menu commands for custom controls, the IDE sends a [reqObeyCommand](#) request. This request specifies the command the user has chosen from the **Edit** menu in the `itemHit` field of the panel parameter block.

In response to the [reqObeyCommand](#) request, plug-ins typically change internal state to reflect an updated selection in the custom control, or copy data to and from the clipboard. When pasting or selecting, it is often necessary to redraw custom items as well. To redraw a custom item, a plug-in can invalidate the item using [CWPanlInvalItem](#).

## Handling Drag And Drop

Previous versions of the IDE supported drag and drop interaction between other applications, such as the Finder, and custom items in preference panels. Custom items were available in panel interfaces constructed from Mac OS 'DITL' resources. The use of traditional Mac OS 'DITL' resources is now deprecated, and plug-in authors are encouraged to use PowerPlant resources instead. See [User Interface Resources](#) for more information.

The following sections document the drag and drop support previously available to 'DITL'-based panels. The IDE still supports these requests for old panels, but this support may be removed in the future. Custom items, and therefore drag and drop, are not currently supported in panels constructed from 'PPob' resources.

### Starting a Drag

The IDE sends the [reqDragEnter](#) request when the user drags an object onto a custom 'DITL'-based dialog item. If the item handles dragging and dropping, the panel should use Drag Manager routines to check the kinds of objects being dragged and highlight the item. Use the `dragref` field in the panel parameter block to get the Drag Manager reference value for the current drag operation.

To make sure that scroll bars and other controls aren't part of the drag destination, adjust the values in `dragrect`, in the panel parameter block. To support autoscrolling, adjust `dragrect` to expand the drag destination vertically.

The panel indicates to the IDE that a drag and drop operation does not apply to an item by returning the `paramErr` error code. Panels can reject drag operations which contain no data of the appropriate type by returning the `dragNotAcceptedErr` error. Panels accept drags by returning `cwNoErr` (or the Mac OS error code `noErr`, which has the same numeric value).

**WARNING!**

While handling a [reqDragEnter](#), [reqDragWithin](#), [reqDragExit](#), or [reqDragDrop](#) request, a panel should not call `WaitNextEvent` or any other Mac OS routine that might result in a process switch. Switching processes during a drag operation will crash the computer.

---

For more information on the Drag Manager, refer to Apple's *Drag Manager Programmer's Guide*. For information on user interface details, refer to Apple's Drag and Drop *Human Interface Guidelines*.

### Requests While Dragging

If the panel accepts a [reqDragEnter](#) request for a dialog item, the IDE then sends [reqDragWithin](#) requests. The IDE sends this request continually until the user either drags the object out of the dialog item or drops the object onto the item.

Typically, plug-ins either take no action in response to [reqDragWithin](#) requests, or they automatically scroll their contents, affording the user control over where an item is dropped within the target item. For an example of this, see the **Access Paths** panel (drag a file from the Finder to a path list).

### Exiting a Drag

If the user drags the object out of the dialog item, the IDE sends the [reqDragExit](#) request. The panel should erase the item's highlighting when it receives this request.

### Ending a Drag

If the user drops the object onto the dialog item, the IDE sends a [reqDragDrop](#) request. Optionally, the panel can tell the IDE to send a [reqItemHit](#) request for any of its items by setting `itemHit` before returning to the IDE.

## Handling Apple Events

Panel plug-ins which provide an '`aete`' resource will receive Apple Event requests from the IDE. The requests will be sent when a script inspects or changes the value of a panel setting.

## Getting data through Apple Events

The IDE sends the [reqAEGetPref](#) request when it receives an Apple Event to get information from the panel. The IDE passes the Apple Event information and the data to use in the panel parameter block. The Apple Event keyword is passed in the `prefsKeyword` variable. The Apple Event data is returned to the IDE in the `prefsDesc` variable. The data to extract the information from is passed in `currentPrefs`.

The IDE sends the [reqAEGetPref](#) request when it receives an Apple Event to get panel information. The IDE passes information about the Apple Event, as well as the panel's settings data, in the panel parameter block.

The Apple Event keyword is passed in the `prefsKeyword` variable. The Apple Event data containing the value of the specified setting is returned by the plug-in in the `prefsDesc` variable. The settings data to extract the information from is passed in `currentPrefs`.

A plug-in examines `prefsKeyword` to determine which setting value to return to the IDE. It then extracts the value of the setting from the `currentPrefs` handle, and stores it in the `prefsDesc` Apple Event descriptor.

---

**NOTE** The plug-in is responsible for allocating the Apple Event descriptor using the Mac OS toolbox routine `AECreatDesc( )`.

---

## Setting data through Apple Events

The IDE sends the [reqAESetPref](#) request when it receives an Apple Event to set panel information. The IDE passes information about the Apple Event, as well as the panel's settings data, in the panel parameter block.

The Apple Event keyword is passed in the `prefsKeyword` variable. The Apple Event data is passed in the `prefsDesc` variable. The settings data to modify is passed in `currentPrefs`.

A plug-in examines `prefsKeyword` to determine which setting to change. It then extracts the new value for the setting from the

`prefsDesc` Apple Event descriptor, and stores it in the appropriate setting in the `currentPrefs` handle.

---

**NOTE** The plug-in is responsible for disposing the Apple Event descriptor using the Mac OS toolbox routine `AEDisposeDesc()`.

---

# Settings Panel Plug-in API Reference

---

This chapter describes the plug-in API services available to settings panel plug-ins.

## Settings Panel Plug-in API Reference Overview

The reference material in this chapter is organized into the following sections:

- [Routines for Settings Panel Plug-ins](#)
- [User Routines for Settings Panel Plug-ins](#)
- [Data Structures for Settings Panel Plug-ins](#)
- [Constants for Settings Panel Plug-ins](#)
- [Settings Panel Result Codes](#)

**NOTE**

In many places, this chapter refers to the **Target Settings** dialog box. In most instances, this is a simplification for the sake of brevity. In fact, panel plug-ins can appear in any of three dialogs: the **Edit > Preferences** dialog, the **Edit > Target Settings** dialog, and the **Edit > Version Control Settings** dialog. This list is shortened to just **Target Settings**, to clarify the text, and because most panel plug-ins appear in this dialog.

---

## Routines for Settings Panel Plug-ins

This section describes all routines available to settings panel plug-ins.

## Organization of Function Reference

Because the API routines for both Windows and Mac OS plug-ins differ in declaration but have similar function, this section of the reference is organized slightly differently from other reference sections.

The organizational differences directly reflect differences in the Windows and Mac OS versions of the settings panel plug-in API. These differences include the following:

1. Windows routines, except for XML routines, start with a “`CWPanel`” prefix. Windows XML routines start with just a “`CW`” prefix. All Mac OS routines start with a “`CWPanl`” prefix.
2. The plug-in state parameter passed to all routines (`context`) is of type [`CWPPluginContext`](#) on Windows and type [`PanelParamBlkPtr`](#) on Mac OS.
3. Windows routines use handles of type [`CWMemHandle`](#). Mac OS routines use Mac OS toolbox handles of type `Handle`.

Because similarly named Windows and Mac OS routines are functionally identical, the Mac OS routine entries simply refer to their Windows counterparts.

---

**NOTE** `DropInPanel.h` (which was for the Mac OS) and `DropInPanelWin32.h` have been deprecated. A cross-platform header file, `CWDropInPanel.h`, has superceded the header files. The deprecated header files are still valid, for backwards compatibility.

---

---

**NOTE** In the future, the XML routines will be moved from the panel API to the core plug-in API.

---

## Plug-in Context

All settings panel routines provided by the CodeWarrior IDE require a value of type [`CWPPluginContext`](#) (on Windows) or of type [`PanelParamBlkPtr`](#) (on Mac OS) to be passed as the first

parameter. To avoid duplication, this parameter is only explained once in detail, rather than for every routine.

- Windows See “[CWPluginContext](#)” on page 172 and “[Main Entry Point context Parameter](#)” on page 77 for more information.
- Mac OS See “[PanelParamBlkPtr](#)” on page 485 and “[PanelParameterBlock](#)” on page 485 for more information.

## Alphabetical Routine Index

This section lists all settings panel routines alphabetically.

- [CWGetArraySettingElement](#)
- [CWGetArraySettingSize](#)
- [CWGetBooleanValue](#)
- [CWGetFloatingPointValue](#)
- [CWGetIntegerValue](#)
- [CWGetNamedSetting](#)
- [CWGetRelativePathValue](#)
- [CWGetStringValue](#)
- [CWGetStructureSettingField](#)
- [CWPanelActivateItem](#)
- [CWPanelAppendItems](#)
- [CWPanelChooseRelativePath](#)
- [CWPanelDeleteListItem](#)
- [CWPanelEnableItem](#)
- [CWPanelGetCurrentPrefs](#)
- [CWPanelGetDebugFlag](#)
- [CWPanelGetDialogItemHit](#)
- [CWPanelGetFactoryPrefs](#)
- [CWPanelGetItemData](#)
- [CWPanelGetItemMaxLength](#)
- [CWPanelGetItemText](#)
- [CWPanelGetItemTextHandle](#)

## Settings Panel Plug-in API Reference

### Alphabetical Routine Index

---

- [CWPanelGetItemValue](#)
- [CWPanelGetListItemText](#)
- [CWPanelGetNumBaseDialogItems](#)
- [CWPanelGetOriginalPrefs](#)
- [CWPanelGetPanelPrefs](#)
- [CWPanelGetRelativePathString](#)
- [CWPanelInsertListItem](#)
- [CWPanelInvalItem](#)
- [CWPanelSetFactoryFlag](#)
- [CWPanelSetItemData](#)
- [CWPanelSetItemMaxLength](#)
- [CWPanelSetItemText](#)
- [CWPanelSetItemTextHandle](#)
- [CWPanelSetItemValue](#)
- [CWPanelSetListItemText](#)
- [CWPanelSetRecompileFlag](#)
- [CWPanelSetRelinkFlag](#)
- [CWPanelSetReparseFlag](#)
- [CWPanelSetResetPathsFlag](#)
- [CWPanelSetRevertFlag](#)
- [CWPanelShowItem](#)
- [CWPanelValidItem](#)
- [CWPanlActivateItem](#)
- [CWPanlAppendItems](#)
- [CWPanlChooseRelativePath](#)
- [CWPanlDrawPanelBox](#)
- [CWPanlDrawUserItemBox](#)
- [CWPanlEnableItem](#)
- [CWPanlGetArraySettingElement](#)
- [CWPanlGetArraySettingSize](#)
- [CWPanlGetBooleanValue](#)

- [CWPanlGetFloatingPointValue](#)
- [CWPanlGetIntegerValue](#)
- [CWPanlGetItemControl](#)
- [CWPanlGetItemData](#)
- [CWPanlGetItemMaxLength](#)
- [CWPanlGetItemRect](#)
- [CWPanlGetItemText](#)
- [CWPanlGetItemTextHandle](#)
- [CWPanlGetItemValue](#)
- [CWPanlGetMacPort](#)
- [CWPanlGetNamedSetting](#)
- [CWPanlGetPanelPrefs](#)
- [CWPanlGetRelativePathString](#)
- [CWPanlGetRelativePathValue](#)
- [CWPanlGetStringValue](#)
- [CWPanlGetStructureSettingField](#)
- [CWPanlInstallUserItem](#)
- [CWPanlInvalItem](#)
- [CWPanlReadBooleanSetting](#)
- [CWPanlReadFloatingPointSetting](#)
- [CWPanlReadIntegerSetting](#)
- [CWPanlReadRelativePathAEDesc](#)
- [CWPanlReadRelativePathSetting](#)
- [CWPanlReadStringSetting](#)
- [CWPanlRemoveUserItem](#)
- [CWPanlSetBooleanValue](#)
- [CWPanlSetFloatingPointValue](#)
- [CWPanlSetIntegerValue](#)
- [CWPanlSetItemData](#)
- [CWPanlSetItemMaxLength](#)
- [CWPanlSetItemText](#)

- [CWPanlSetItemTextHandle](#)
- [CWPanlSetItemValue](#)
- [CWPanlSetRelativePathValue](#)
- [CWPanlSetStringValue](#)
- [CWPanlShowItem](#)
- [CWPanlValidItem](#)
- [CWPanlWriteBooleanSetting](#)
- [CWPanlWriteFloatingPointSetting](#)
- [CWPanlWriteIntegerSetting](#)
- [CWPanlWriteRelativePathAEDesc](#)
- [CWPanlWriteRelativePathSetting](#)
- [CWPanlWriteStringSetting](#)
- [CWReadBooleanSetting](#)
- [CWReadFloatingPointSetting](#)
- [CWReadIntegerSetting](#)
- [CWReadRelativePathSetting](#)
- [CWReadStringSetting](#)
- [CWSetBooleanValue](#)
- [CWSetFloatingPointValue](#)
- [CWSetIntegerValue](#)
- [CWSetRelativePathValue](#)
- [CWSetStringValue](#)
- [CWWriteBooleanSetting](#)
- [CWWriteFloatingPointSetting](#)
- [CWWriteIntegerSetting](#)
- [CWWriteRelativePathSetting](#)
- [CWWriteStringSetting](#)

## Functional Routine Index

### Manipulating Panel Controls

- [CWPanelAppendItems](#) , [CWPanlAppendItems](#)
-

- [CWPanelActivateItem](#), [CWPanlActivateItem](#)
- [CWPanelEnableItem](#), [CWPanlEnableItem](#)
- [CWPanelGetItemValue](#), [CWPanlGetItemValue](#)
- [CWPanelSetItemValue](#), [CWPanlSetItemValue](#)
- [CWPanelShowItem](#), [CWPanlShowItem](#)
- [CWPanelInvalItem](#), [CWPanlInvalItem](#)
- [CWPanelValidItem](#), [CWPanlValidItem](#)
- [CWPanelGetItemData](#), [CWPanlGetItemData](#)
- [CWPanelSetItemData](#), [CWPanlSetItemData](#)

### **Getting and Setting Text Controls**

- [CWPanelGetItemText](#), [CWPanlGetItemText](#)
- [CWPanelSetItemText](#), [CWPanlSetItemText](#)
- [CWPanelGetItemTextHandle](#),  
[CWPanlGetItemTextHandle](#)
- [CWPanelSetItemTextHandle](#),  
[CWPanlSetItemTextHandle](#)
- [CWPanelGetItemMaxLength](#), [CWPanlGetItemMaxLength](#)
- [CWPanelSetItemMaxLength](#), [CWPanlSetItemMaxLength](#)

### **Obtaining Settings Handles**

- [CWPanelGetCurrentPrefs](#)
- [CWPanelGetOriginalPrefs](#)
- [CWPanelGetFactoryPrefs](#)
- [CWPanelGetPanelPrefs](#), [CWPanlGetPanelPrefs](#)

### **Handling Panel Events (Windows)**

- [CWPanelGetDialogItemHit](#)
- [CWPanelGetNumBaseDialogItems](#)

### **Manipulating Combo Box Item Lists (Windows)**

- [CWPanelDeleteListItem](#)
- [CWPanelInsertListItem](#)
- [CWPanelGetListItemText](#)

- [CWPanelSetListItemText](#)

### Obtaining IDE State (Windows)

- [CWPanelGetDebugFlag](#)
- [CWPanelSetFactoryFlag](#)
- [CWPanelSetRecompileFlag](#)
- [CWPanelSetRelinkFlag](#)
- [CWPanelSetReparseFlag](#)
- [CWPanelSetResetPathsFlag](#)
- [CWPanelSetRevertFlag](#)

### Manipulating Controls (Mac OS)

- [CWPanlGetItemControl](#)
- [CWPanlGetItemRect](#)
- [CWPanlGetMacPort](#)

### Custom Controls (User Items) (Mac OS)

- [CWPanlInstallUserItem](#)
- [CWPanlRemoveUserItem](#)
- [CWPanlDrawPanelBox](#)
- [CWPanlDrawUserItemBox](#)

### Manipulating Relative Paths

- [CWPanelChooseRelativePath](#),  
[CWPanlChooseRelativePath](#)
- [CWPanelGetRelativePathString](#),  
[CWPanlGetRelativePathString](#)
- [CWPanlReadRelativePathAEDesc](#)
- [CWPanlWriteRelativePathAEDesc](#)

### Reading Primitive XML Types

- [CWReadBooleanSetting](#), [CWPanlReadBooleanSetting](#)
- [CWReadFloatingPointSetting](#),  
[CWPanlReadFloatingPointSetting](#)
- [CWReadIntegerSetting](#), [CWPanlReadIntegerSetting](#)

- [CWReadRelativePathSetting](#),  
[CWPanlReadRelativePathSetting](#)
- [CWReadStringSetting](#), [CWPanlReadStringSetting](#)

### **Writing Primitive XML Types**

- [CWWriteBooleanSetting](#),  
[CWPanlWriteBooleanSetting](#)
- [CWWriteFloatingPointSetting](#),  
[CWPanlWriteFloatingPointSetting](#)
- [CWWriteIntegerSetting](#),  
[CWPanlWriteIntegerSetting](#)
- [CWWriteRelativePathSetting](#),  
[CWPanlWriteRelativePathSetting](#)
- [CWWriteStringSetting](#), [CWPanlWriteStringSetting](#)

### **Reading and Writing XML Structures and Arrays**

- [CWGetNamedSetting](#), [CWPanlGetNamedSetting](#)
- [CWGetStructureSettingField](#),  
[CWPanlGetStructureSettingField](#)
- [CWGetArraySettingElement](#),  
[CWPanlGetArraySettingElement](#)
- [CWGetArraySettingSize](#),  
[CWPanlGetArraySettingSize](#)

### **Reading XML Structure and Array Elements**

- [CWGetBooleanValue](#), [CWPanlGetBooleanValue](#)
- [CWGetFloatingPointValue](#),  
[CWPanlGetFloatingPointValue](#)
- [CWGetIntegerValue](#), [CWPanlGetIntegerValue](#)
- [CWGetRelativePathValue](#),  
[CWPanlGetRelativePathValue](#)
- [CWGetStringValue](#), [CWPanlGetStringValue](#)

### **Writing XML Structure and Array Elements**

- [CWSetBooleanValue](#), [CWPanlSetBooleanValue](#)
- [CWSetFloatingPointValue](#),  
[CWPanlSetFloatingPointValue](#)

- [CWSetIntegerValue](#), [CWPanlSetIntegerValue](#)
- [CWSetRelativePathValue](#),  
[CWPanlSetRelativePathValue](#)
- [CWSetStringValue](#), [CWPanlSetStringValue](#)

## CWGetArraySettingElement

Description Returns one element of an XML array, by ID.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWGetArraySettingElement(
    CWPluginContext context,
    CWSettingID settingID,
    long index,
    CWSettingID\* elementSettingID);
```

Parameters Parameters for this function are:

context	Private, opaque IDE state.
settingID	Specifies the XML array containing the element to be returned.
index	A zero-based index specifying the item in the array to be returned.
elementSettingID	Returns the ID of the indexed array element, which may be simple type, or a structured type.

`CWGetArraySettingElement` returns an ID for a field of an XML structure. `CWGetArraySettingElement` behaves differently depending upon when it is called.

During a [reqReadSettings](#) request, this routine will return an error if a field having the specified name is not found. If the named field is found, the ID returned may be used to call one of the following routines to obtain the value of the field:

- [CWGetBooleanValue](#)
- [CWGetFloatingPointValue](#)
- [CWGetIntegerValue](#)
- [CWGetRelativePathValue](#)

- [CWGetStringValue](#)

If the field is another array, the plug-in should call [CWGetArraySettingSize](#) and [CWGetArraySettingElement](#) to read the elements of the array. If the field is a structure, the plug-in should instead call [CWGetStructureSettingField](#) to obtain the fields of the structure.

During a [reqReadSettings](#) request, to get the ID of a top-level XML array setting (that is, a setting whose parent container is implicitly the plug-in's XML stream), a plug-in calls [CWGetNamedSetting](#). The ID returned can be used to call [CWGetArraySettingElement](#).

During a [reqWriteSettings](#) request, this routine will create a new XML setting have the specified name if one has not already been created, and return its ID. The ID returned may be used to call one of the following routines to set the value of the field

- [CWSetBooleanValue](#)
- [CWSetFloatingPointValue](#)
- [CWSetIntegerValue](#)
- [CWSetRelativePathValue](#)
- [CWSetStringValue](#)

If the field to be written is instead another structure or array, the plug-in should call [CWGetStructureSettingField](#) or [CWGetArraySettingElement](#) to create a new setting. Both routines return an ID, which can be used in further calls to any of the above routines.

A plug-in creates a new top-level XML structure setting during a [reqWriteSettings](#) request by calling [CWGetNamedSetting](#). The ID returned can be used to call [CWGetArraySettingElement](#).

See Also

- [“CWGetStructureSettingField” on page 416](#)
- [“CWGetArraySettingSize” on page 410](#)
- [“CWGetNamedSetting” on page 413](#)
- [“CWGetBooleanValue” on page 411](#)

[“CWGetFloatingPointValue” on page 411](#)

[“CWGetIntegerValue” on page 412](#)

[“CWGetRelativePathValue” on page 414](#)

[“CWGetStringValue” on page 415](#)

[“CWSetBooleanValue” on page 471](#)

[“CWSetFloatingPointValue” on page 472](#)

[“CWSetIntegerValue” on page 474](#)

[“CWSetRelativePathValue” on page 475](#)

[“CWSetStringValue” on page 476](#)

## **CWGetArraySettingSize**

**Description** Returns the number of elements in an XML array.

**Prototype**

```
#include <CWDropInPanel.h>
CW_CALLBACK CWGetArraySettingSize(
    CWPluginContext context,
    CWSettingID settingID,
    long* size);
```

**Parameters** Parameters for this function are:

**context** Private, opaque IDE state.

**settingID** Specifies the XML array containing the element to be returned.

**size** Returns the number of elements in the specified array.

**Remarks** `CWGetArraySettingSize` returns the number of elements in an XML array. This is useful when reading in an array of unknown size, or to verify that the size of an array is correct prior to accessing its elements using [CWGetArraySettingElement](#).

**See Also** [“CWGetArraySettingElement” on page 408](#)

## CWGetBooleanValue

Description	Returns the value of a boolean element of an XML array or structure.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWGetBooleanValue( <a href="#">CWPluginContext</a> context, <a href="#">CWSettingID</a> settingID, Boolean* value);
Parameters	Parameters for this function are:  context              Private, opaque IDE state. settingID            Specifies the XML setting for which to return a value. value                 Returns the boolean XML setting value.
Remarks	CWGetBooleanValue returns the value of a boolean XML setting given its ID. Returns an error if the given setting is not of boolean type.  This routine is typically used to read a structure field or array element obtained by calling <a href="#">CWGetStructureSettingField</a> or <a href="#">CWGetArraySettingElement</a> . To read the value of a top-level boolean setting, use <a href="#">CWReadBooleanSetting</a> instead.
See Also	<a href="#">“CWGetStructureSettingField” on page 416</a> <a href="#">“CWGetArraySettingElement” on page 408</a> <a href="#">“CWReadBooleanSetting” on page 467</a>

## CWGetFloatingPointValue

Description	Returns the value of a floating point element of an XML array or structure.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWGetFloatingPointValue( <a href="#">CWPluginContext</a> context, <a href="#">CWSettingID</a> settingID, double* value);

---

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	settingID    Specifies the XML setting for which to return a value.
	value        Returns the double-precision floating point XML setting value.
Remarks	CWGetFloatingPointValue returns the value of a floating point XML setting given its ID. Returns an error if the given setting is not of floating point type.  This routine is typically used to read a structure field or array element obtained by calling <a href="#">CWGetStructureSettingField</a> or <a href="#">CWGetArraySettingElement</a> . To read the value of a top-level floating point setting, use <a href="#">CWReadFloatingPointSetting</a> instead.
See Also	<a href="#">“CWGetStructureSettingField” on page 416</a> <a href="#">“CWGetArraySettingElement” on page 408</a> <a href="#">“CWReadFloatingPointSetting” on page 468</a>

## **CWGetIntegerValue**

Description      Returns the value of an integer element of an XML array or structure.

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWGetIntegerValue(
    CWPluginContext context,
    CWSettingID settingID,
    long* value);
```

Parameters     Parameters for this function are:

context	Private, opaque IDE state.
settingID	Specifies the XML setting for which to return a value.
value	Returns the long integer XML setting value.

Remarks CWGetIntegerValue returns the value of an integer XML setting given its ID. Returns an error if the given setting is not of integer type.

This routine is typically used to read a structure field or array element obtained by calling [CWGetStructureSettingField](#) or [CWGetArraySettingElement](#). To read the value of a top-level integer setting, use [CWReadIntegerSetting](#) instead.

See Also [“CWGetStructureSettingField” on page 416](#)

[“CWGetArraySettingElement” on page 408](#)

[“CWReadIntegerSetting” on page 469](#)

## CWGetNamedSetting

Description Returns the ID of a top-level XML setting specified by name.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWGetNamedSetting(
    CWPluginContext context,
    const char* name,
    CWSettingID* settingID);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

name Specifies the name of an XML setting, as a C string.

settingID Returns the ID of a newly-created or existing setting container.

Remarks CWGetNamedSetting returns the ID of an XML setting contained in a panel's XML data. Normally, the ID refers to a structured XML setting, such as an array or structure.

The returned ID can be used to read or write a nested structure, using [CWGetStructureSettingField](#). To read or write a nested array, call [CWGetArraySettingElement](#).

---

**NOTE** CWGetNamedSetting can actually be used to get or create an ID for any XML setting type, whether structured or not. However, it is

usually easier to use the `CWRead...` and `CWWrite...` calls when reading non-structured types.

---

A setting returned (while reading) or created (during writing) by `CWGetNamedSetting` is always a top-level setting, meaning that the parent container is the plug-in's XML stream (either input or output).

`CWGetNamedSetting` behaves differently during a read request than during a write request. During a [`reqReadSettings`](#) request, `CWGetNamedSetting` will return an error if the named setting is not found in the XML input stream. During a [`reqWriteSettings`](#) request, `CWGetNamedSetting` will instead create a top-level setting having the specified name.

See Also ["CWGetStructureSettingField" on page 416](#)

["CWGetArraySettingElement" on page 408](#)

["reqReadSettings" on page 512](#)

["reqWriteSettings" on page 517](#)

## CWGetRelativePathValue

Description Returns the value of a relative path element of an XML array or structure.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWGetRelativePathValue(
    CWPluginContext context,
    CWSettingID settingID,
    CWRelativePath* value);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

settingID Specifies the XML setting for which to return a value.

value Returns the relative path XML setting value.

Remarks	CWGetBooleanValue returns the value of a relative path XML setting given its ID. Returns an error if the given setting is not a relative path.
	This routine is typically used to read a structure field or array element obtained by calling <a href="#">CWGetStructureSettingField</a> or <a href="#">CWGetArraySettingElement</a> . To read the value of a top-level relative path setting, use <a href="#">CWReadRelativePathSetting</a> instead.
See Also	<a href="#">“CWGetStructureSettingField” on page 416</a> <a href="#">“CWGetArraySettingElement” on page 408</a> <a href="#">“CWReadRelativePathSetting” on page 470</a>

## CWGetStringValue

Description	Returns the value of a string element of an XML array or structure.						
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWGetStringValue(     <a href="#">CWPluginContext</a>    context,     <a href="#">CWSettingID</a>        settingID,     const char**          value);</pre>						
Parameters	<p>Parameters for this function are:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;"><b>context</b></td><td>Private, opaque IDE state.</td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><b>settingID</b></td><td>Specifies the XML setting for which to return a value.</td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><b>value</b></td><td>Returns a C string containing the contents of an XML string setting. The string remains the property of the IDE.</td></tr> </table>	<b>context</b>	Private, opaque IDE state.	<b>settingID</b>	Specifies the XML setting for which to return a value.	<b>value</b>	Returns a C string containing the contents of an XML string setting. The string remains the property of the IDE.
<b>context</b>	Private, opaque IDE state.						
<b>settingID</b>	Specifies the XML setting for which to return a value.						
<b>value</b>	Returns a C string containing the contents of an XML string setting. The string remains the property of the IDE.						
Remarks	CWGetStringValue returns the value of a string XML setting given its ID. It returns an error if the given setting is not of string type.						
<b>WARNING!</b>	<p>The <code>value</code> string returned remains the property of the IDE, and may be deallocated by the IDE at the end of the pending request. Therefore, to retain the text returned in <code>value</code> globally, be sure to copy it into another string before returning to the IDE.</p>						

This routine is typically used to read a structure field or array element obtained by calling [CWGetStructureSettingField](#) or [CWGetArraySettingElement](#). To read the value of a top-level string setting, use [CWReadStringSetting](#) instead.

See Also [“CWGetStructureSettingField” on page 416](#)

[“CWGetArraySettingElement” on page 408](#)

[“CWReadStringSetting” on page 470](#)

## CWGetStructureSettingField

Description Returns one field of an XML structure, by ID.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWGetStructureSettingField(
    CWPluginContext context,
    CWSettingID settingID,
    const char* name,
    CWSettingID* fieldSettingID);
```

Parameters Parameters for this function are:

- |                |  |
|----------------|--|
| context        | Private, opaque IDE state.                           |
| settingID      | Specifies the IDE of an XML structure.               |
| name           | Specifies the name of the structure field to return. |
| fieldSettingID | Returns the ID of the named field.                   |

Remarks CWGetStructureSettingField returns an ID for a field of an XML structure. CWGetStructureSettingField behaves differently depending upon when it is called.

During a [reqReadSettings](#) request, this routine will return an error if a field having the specified name is not found. If the named field is found, the ID returned may be used to call one of the following routines to obtain the value of the field:

- [CWGetBooleanValue](#)
- [CWGetFloatingPointValue](#)
- [CWGetIntegerValue](#)

- [CWGetRelativePathValue](#)
- [CWGetStringValue](#)

If the field is another structure, the plug-in should instead call `CWGetStructureSettingField` to obtain the fields of the structure. If the field is an array, the plug-in should call [CWGetArraySettingSize](#) and [CWGetArraySettingElement](#) to read the elements of the array.

During a [reqReadSettings](#) request, to get the ID of a top-level XML structure setting (that is, a setting whose parent container is implicitly the plug-in's XML stream), the plug-in calls [CWGetNamedSetting](#). The ID returned can be used to call `CWGetStructureSettingField`.

During a [reqWriteSettings](#) request, this routine will create a new XML setting have the specified name if one has not already been created, and return its ID. The ID returned may be used to call one of the following routines to set the value of the field

- [CWSetBooleanValue](#)
- [CWSetFloatingPointValue](#)
- [CWSetIntegerValue](#)
- [CWSetRelativePathValue](#)
- [CWSetStringValue](#)

If the field to be written is instead another structure or array, the plug-in should call `CWGetStructureSettingField` or [CWGetArraySettingElement](#) to create a new setting. Both routines return an ID, which can be used in further calls to any of the above routines.

A plug-in creates a new top-level XML structure setting during a [reqWriteSettings](#) request by calling [CWGetNamedSetting](#). The ID returned can be used to call `CWGetStructureSettingField`.

- See Also
- [“CWGetArraySettingElement” on page 408](#)
  - [“CWGetArraySettingSize” on page 410](#)
  - [“CWGetNamedSetting” on page 413](#)

[“reqReadSettings” on page 512](#)

[“reqWriteSettings” on page 517](#)

[“CWGetBooleanValue” on page 411](#)

[“CWGetFloatingPointValue” on page 411](#)

[“CWGetIntegerValue” on page 412](#)

[“CWGetRelativePathValue” on page 414](#)

[“CWGetStringValue” on page 415](#)

[“CWSetBooleanValue” on page 471](#)

[“CWSetFloatingPointValue” on page 472](#)

[“CWSetIntegerValue” on page 474](#)

[“CWSetRelativePathValue” on page 475](#)

[“CWSetStringValue” on page 476](#)

## **CWPanelActivateItem**

Description      Switches user input focus to the specified item.

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelActivateItem(
    CWPluginContext context,
    long whichItem);
```

Parameters    Parameters for this function are:

context        Private, opaque IDE state.

whichItem      The plug-in specifies the item to give the input focus.

Remarks       A settings panel plug-in calls this routine to make the specified item the focus of user input. For example, to switch user input to a text edit box, call `CWPanelActivateItem` with the ID of the edit box.

**TIP** Generally, the user should be in control of input focus; use this routine only in special cases where appropriate.

---

See Also [“CWPanelEnableItem” on page 423](#)

[“Managing Input Focus” on page 373](#)

## CWPanelAppendItems

Description Adds the panel's dialog items to the **Target Settings** dialog box.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelAppendItems(
    CWPluginContext context,
    short ditlID);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

ditlID The plug-in specifies the resource ID of the dialog resource (on Windows) or a 'PPop' resource (on Mac OS) to add to the **Target Settings** dialog box.

Remarks Plug-ins call `CWPanelAppendItems` to create their user interfaces. `CWPanelAppendItems` constructs the panel's user interface controls from a dialog resource specified by ID. A panel's controls are added to the dialog specified by the `panelScope` field of a plug-in's [PanelFlags Structure](#).

The IDE expects to find a dialog resource among the currently accessible resources. On Windows, the dialog resource should be linked into the plug-in DLL. On Mac OS, the dialog ('PPop') resource should be copied into the plug-in's resource fork.

A settings panel plug-in should call `CWPanelAppendItems` in response to a [reqInitDialog](#) request. After the panel's user interface has been created, the plug-in can make any additional changes needed to its interface.

Windows On Windows, plug-ins may need to call [CWPanelGetListListItemText](#), [CWPanelSetListListItemText](#),

[CWPanelInsertListItem](#), and [CWPanelDeleteListItem](#) to set up the items in a combo-box list.

**See Also**

[“reqInitDialog” on page 509](#)

[“reqInitiDialog Request” on page 354](#)

[“CWPanelGetListItemText” on page 432](#)

[“CWPanelSetListItemText” on page 445](#)

[“CWPanelDeleteListItem” on page 422](#)

[“CWPanelInsertListItem” on page 438](#)

## CWPanelChooseRelativePath

**Description** Prompts the user to select a folder, and returns the specified folder as a relative path.

**Prototype**

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelChooseRelativePath(
    CWPluginContext context,
    CWRelativePath* ioPath,
    Boolean isFolder,
    short filterCount,
    void* filterList,
    char* prompt);
```

**Parameters** Parameters for this function are:

**context** Private, opaque IDE state.

**ioPath** Specifies the initial location for selecting a file or folder, and returns the file or folder chosen by the user, as a relative path.

**isFolder** Specifies whether a file or folder is to be selected. Selects a folder if true.

**filterCount** Specifies the number of types used to filter the file list, when displaying files. Used only on Mac OS.

filterList	On Windows, this specifies a file or directory filter string.  On Mac OS, this points to an <code>SFTypelist</code> (an array of 4-character <code>OSType</code> file type codes).
prompt	Specifies the user prompt to display when selecting a file or folder. This parameter is currently unused and ignored. Specified as a C string on Windows, and a Pascal string on Mac OS.
Remarks	<p>Plug-ins call <code>CWPanelChooseRelativePath</code> to prompt the user to select a file or folder, specified relatively using a <a href="#"><code>CWRelativePath</code></a> structure. Relative path specifications are useful for tracking files when moving project files.</p> <p><code>CWPanelChooseRelativePath</code> prompts the user to choose a file or folder, according to the value of <code>isFolder</code>, starting in the location specified by <code>ioPath</code>. It returns a relative path specification for the file or folder chosen in <code>ioPath</code>. If the user cancels the dialog, <code>CWPanelChooseRelativePath</code> returns <code>cwErrUserCanceled</code>.</p> <p>Currently, the text of <code>prompt</code> is ignored.</p> <p><code>CWPanelChooseRelativePath</code> filters the list of files (when selecting files) using <code>filterList</code>. <code>filterList</code> controls which files appear for selection. The format of the parameter is different on Windows and Mac OS.</p>
Windows	<p>On Windows, the <code>filterList</code> parameter is formatted the same way the <code>lpstrFilter</code> field of an <code>OPENFILENAME</code> structure passed to the <code>GetOpenFileName</code> common dialog call is formatted. The <code>filterCount</code> parameter is ignored.</p> <p>Briefly, the filter string consists of pairs of null-terminated strings, concatenated (but with intervening NULL characters intact) into one large string, terminated by a final NULL character. Each pair of strings consists of a filter name, followed by a pattern specification.</p> <p>The pattern specification can include legal file name characters and the DOS “*” wildcard character, and usually specifies the literal</p>

value of a file extension. Multiple file patterns can be included in one string by separating each with a semicolon (“;”) character.

For further details, see Microsoft’s MSDN developer information.

Mac OS      On Mac OS, the list of displayed files is filtered against a standard SFTypelist array of four-character codes, listing the file types to display. The filterCount parameter specifies the number of files in the type list. To display all files, specify a count of -1.

---

**NOTE** These comments apply to the Mac OS version of this routine, named [CWPanelChooseRelativePath](#).

---

See Also [“CWPanelGetRelativePathString” on page 436](#)

[“CWRelativePath” on page 178](#)

[“CWRelativePathTypes” on page 181](#)

[“CWResolveRelativePath” on page 142](#)

## CWPanelDeleteListltem

Description Deletes an item from a Windows combo-box list.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelDeleteListltem(
    CWPluginContext context,
    long             dlgItemID,
    long             index);
```

Parameters Parameters for this function are:

context      Private, opaque IDE state.

dlgItemID     Specifies the ID of the combo-box containing the item to delete.

index        A one-based index that specifies the item to delete from a combo-box list.

Remarks CWPanelDeleteListltem removes one item from a comb-box list. Use CWPanelDeleteListltem to maintain the list of items appearing in a combo-box.

---

For example, if your settings panel maintains a fixed-length history of recent text entries in a combo-box, you can use `CWPanelDeleteListItem` to delete old items from the history and [`CWPanelInsertListItem`](#) to add new ones to it.

See Also    ["CWPanelInsertListItem" on page 438](#)

## **CWPanelEnableItem**

Description	Enables or disables a panel control.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelEnableItem(     <a href="#"><u>CWPluginContext</u></a> context,     long whichItem,     Boolean enableIt);</pre>
Parameters	Parameters for this function are:
	context              Private, opaque IDE state.
	whichItem            Specifies the ID of the panel control to enable or disable.
	enableIt            Specifies whether to enable the control.
Remarks	A plug-in calls this routine to enable or disable a panel item. Enabled controls respond to events, and are drawn normally. Disabled items are dimmed, and do not respond to user actions.  <code>CWPanelEnableItem</code> is typically used to enable or disable one control based upon the state of another. For example, a panel for a linker capable of producing relocatable executables and absolute address executables could use <code>CWPanelEnableItem</code> to enable and disable the absolute load address edit box depending upon whether the selected executable type is relative or absolute.

See Also    ["CWPanelActivateItem" on page 418](#)

["CWPanelShowItem" on page 450](#)

## **CWPanelGetCurrentPrefs**

Description	Returns a handle containing the panel's current settings data.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt;</pre>

## Settings Panel Plug-in API Reference

### CWPanelGetDebugFlag

---

```
CW_CALLBACK CWPanelGetCurrentPrefs(  
    CWPluginContext context,  
    CWMemHandle* currentPrefs);
```

Parameters	Parameters for this function are:
	context              Private, opaque IDE state.
	currentPrefs        Returns a handle containing the plug-in's current settings data.
Remarks	A plug-in calls <code>CWPanelGetCurrentPrefs</code> to obtain the IDE's current copy of its settings data. The handle returned remains the IDE's property. A plug-in directly modifies the handle's contents to save its current settings data.  Before accessing the returned settings handle, a plug-in should call <code>CWLockMemHandle</code> to lock the handle. When finished accessing the handle, a plug-in should call <code>CWUnlockMemHandle</code> . To determine the size of the handle, use <code>CWGetMemHandleSize</code> . To resize the handle, use <code>CWResizeMemHandle</code> .
Mac OS	On Mac OS, plug-ins should instead directly access the <code>factoryPrefs</code> field of the <code>PanelParameterBlock</code> . The handle should be locked, unlocked, and resized using the Mac OS toolbox calls <code>HLock</code> , <code>HUnlock</code> , and <code>GetHandleSize</code> and <code>SetHandleSize</code> .
See Also	<a href="#">“CWPanelGetFactoryPrefs” on page 426</a> <a href="#">“CWPanelGetOriginalPrefs” on page 434</a> <a href="#">“CWPanelGetPanelPrefs” on page 435</a> <a href="#">“Settings Panel Data” on page 342</a>

## CWPanelGetDebugFlag

Description     Indicates whether debugging is enabled for the current target.

```
#include <CWDropInPanel.h>  
CW_CALLBACK CWPanelGetDebugFlag(  
    CWPluginContext context,  
    Boolean* debugOn);
```

Parameters	Parameters for this function are:
	context         Private, opaque IDE state.
	debugOn        Returns true if debugging is currently on, and false otherwise.
Remarks	CWPanelGetDebugFlag returns true if the user has enabled debugging by selecting <b>Project &gt; Enable Debugger</b> .
<b>NOTE</b>	This flag should not be confused with the value returned by <a href="#">CWIsGeneratingDebugInfo</a> . CWPanelGetDebugFlag indicates whether debugging of the final executable is enabled. The value returned by <a href="#">CWIsGeneratingDebugInfo</a> indicates whether debugging information is enabled for a particular project file.
	Plug-ins typically call CWPanelGetDebugFlag in response to a <a href="#">reqSetupDebug</a> request, to determine whether debugging has been enabled or disabled, and to then make any necessary modifications to settings data.
See Also	<a href="#">"reqSetupDebug" on page 513</a> <a href="#">"reqSetupDebug Request" on page 364</a>

## **CWPanelGetDialogItemHit**

Description	Returns the ID of the most-recently operated panel control.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelGetDialogItemHit(     <a href="#">CWPluginContext</a>    context,     short*                 itemHit);</pre>
Parameters	Parameters for this function are:
	context         Private, opaque IDE state.
	itemHit        Returns the ID of the item hit during a <a href="#">reqItemHit</a> request.
Remarks	CWPanelGetDialogItemHit returns the ID of the control the user most recently interacted with. In response to a <a href="#">reqItemHit</a>

request, plug-ins call `CWPanelGetDialogItemHit` to determine which item was hit.

The item number returned by `CWPanelGetDialogItemHit` must be adjusted by subtracting the base number of items appearing in the dialog, prior to the addition of the panel's controls. Use [`CWPanelGetNumBaseDialogItems`](#) to get this count.

Depending upon the control that was hit, plug-ins may wish to take further action, such as enabling, disabling, and changing the values of other controls. For example, a preference panel might provide a button for selecting a folder. When the plug-in receives an [`reqItemHit`](#) request, it checks to see if the ID of the item hit matches the ID of the button, and if so calls [`CWPanelChooseRelativePath`](#) to select a folder.

Mac OS On Mac OS, plug-ins instead examine the `itemHit` and `baseItems` fields of the [`PanelParameterBlock`](#) to determine the item hit and the base number of dialog items.

See Also ["CWPanelGetNumBaseDialogItems" on page 433](#)

["Determining the "Hit" Dialog Item" on page 350](#)

["reqItemHit" on page 510](#)

["reqItemHit Request" on page 365](#)

## **CWPanelGetFactoryPrefs**

Description Returns a handle containing the panel's factory default settings data.

Prototype `#include <CWDropInPanel.h>`  
`CW_CALLBACK CWPanelGetFactoryPrefs(`  
`CWPluginContext     context,`  
`CWMemHandle*     factoryPrefs );`

Parameters Parameters for this function are:

context Private, opaque IDE state.

factoryPrefs Returns a handle containing the plug-in's factory default settings data.

Remarks	A plug-in calls <code>CWPanelGetFactoryPrefs</code> to obtain the IDE's current copy of its settings data. The handle returned remains the IDE's property. A plug-in directly modifies the handle's contents to change its factory default settings data.  The factory default settings data is used by the IDE when the user clicks the <b>Factory Settings</b> button in the <b>Target Settings</b> dialog box. Plug-ins should also compare their current settings to their current factory defaults in response to a <a href="#">reqItemHit</a> request, to determine how to set the factory flag using <a href="#">CWPanelSetFactoryFlag</a> .  Before accessing the returned settings handle, a plug-in should call <a href="#">CWLockMemHandle</a> to lock the handle. When finished accessing the handle, a plug-in should call <a href="#">CWUnlockMemHandle</a> . To determine the size of the handle, use <a href="#">CWGetMemHandleSize</a> . To resize the handle, use <a href="#">CWResizeMemHandle</a> .
Mac OS	On Mac OS, plug-ins should instead directly access the <code>factoryPrefs</code> field of the <a href="#">PanelParameterBlock</a> . The handle should be locked, unlocked, and resized using the Mac OS toolbox calls <code>HLock</code> , <code>HUnlock</code> , and <code>GetHandleSize</code> and <code>SetHandleSize</code> .
See Also	<a href="#">“CWPanelGetCurrentPrefs” on page 423</a> <a href="#">“CWPanelGetOriginalPrefs” on page 434</a> <a href="#">“CWPanelGetPanelPrefs” on page 435</a> <a href="#">“Settings Panel Data” on page 342</a>

## CWPanelGetItemData

Description	Returns additional type-specific data for a panel control.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelGetItemData(     <a href="#">CWPluginContext</a>    context,     long                  dlgItemID,     void*                outData,     long*                outDataLength);</pre>
Parameters	Parameters for this function are:

---

context	Private, opaque IDE state.
dlgItemID	Specifies the ID of the item for which to get additional control data.
outData	Points to storage in which to return the additional control information.
outDataLength	Specifies the length of the data to be returned in outData.
Remarks	This routine provides no useful service for third-party plug-in developers at this time. The exact behavior of this routine is subject to change. Plug-ins should not use this routine.
See Also	<a href="#">“CWPanelSetItemData” on page 441</a>

## **CWPanelGetItemMaxLength**

Description	Returns the maximum allowed length for a text item.						
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelGetItemMaxLength(     <a href="#">CWPluginContext</a> context,     long             dlgItemID,     short*          outLength);</pre>						
Parameters	Parameters for this function are:						
	<table><tr><td style="vertical-align: top;">context</td><td>Private, opaque IDE state.</td></tr><tr><td style="vertical-align: top;">dlgItemID</td><td>Specifies the ID of a static text or edit field about which to return length information.</td></tr><tr><td style="vertical-align: top;">outLength</td><td>Returns the maximum length, in characters, allowed when entering text into the item specified by dlgItemID.</td></tr></table>	context	Private, opaque IDE state.	dlgItemID	Specifies the ID of a static text or edit field about which to return length information.	outLength	Returns the maximum length, in characters, allowed when entering text into the item specified by dlgItemID.
context	Private, opaque IDE state.						
dlgItemID	Specifies the ID of a static text or edit field about which to return length information.						
outLength	Returns the maximum length, in characters, allowed when entering text into the item specified by dlgItemID.						
Remarks	To determine the maximum allowed length for text entered into a static text or edit text field, use CWPanelGetItemMaxLength. To set the maximum length, use <a href="#">CWPanelSetItemMaxLength</a> .						
See Also	<a href="#">“CWPanelSetItemMaxLength” on page 441</a> <a href="#">“Validating Input” on page 373</a>						

## CWPanelGetItemText

Description	Returns the text of a dialog item.
Prototype	#include <CWDropInPanel.h>  CW_CALLBACK CWPanelGetItemText( <a href="#">CWPluginContext</a> context, long whichItem, char* str, short maxLen);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. whichItem       Specifies the ID of the item for which to return text. str             Provides space in which the IDE returns the item's text, as a C string on Windows, and a Pascal string on Mac OS. maxLen          Specifies the maximum length of the text to be returned, including the length byte. If the text exceeds this length,
Remarks	This routine returns the text for an editable text field, static text field, or any other type of control's title. Plug-ins commonly call CWPanelGetItemText to extract the text entered into a panel by the user, when handling a <a href="#">reqGetData</a> request.  CWPanelGetItemText is also useful for character string validation. For example, in response to a <a href="#">reqItemHit</a> request, a plug-in might ensure that the characters entered into a text field contain only legal file name characters. To extract the value of a field as a number, a plug-in calls <a href="#">CWPanelGetItemValue</a> instead.  If the text associated with the item is longer than maxLen characters, CWPanelGetItemText returns no error code, but returns as much text as will fit in str. To get the full text of an item, use <a href="#">CWPanelGetItemTextHandle</a> .  To determine the maximum possible length of text entered in a field, call <a href="#">CWPanelGetItemMaxLength</a> . The value returned can be used to allocate space for text returned by CWPanelGetItemText.

See Also [“CWPanelGetItemMaxLength” on page 428](#)

[“CWPanelSetItemText” on page 442](#)

[“CWPanelGetItemTextHandle” on page 430](#)

[“CWPanelGetItemValue” on page 431](#)

[“CWPanelSetValue” on page 444](#)

[“reqGetData” on page 506](#)

[“reqItemHit” on page 510](#)

[“Validating Input” on page 373](#)

## CWPanelGetItemTextHandle

Description Returns a handle containing the text of a panel control.

Prototype #include <CWDropInPanel.h>

```
CW_CALLBACK CWPanelGetItemTextHandle(
    CWPluginContext context,
    long whichItem,
    CWMemHandle* text);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

whichItem Specifies the ID of the item for which to return a text handle.

text Returns a handle containing the text of the item. The handle becomes the caller's property.

Remarks CWPanelGetItemTextHandle returns the text of a static text or edit text item, or the title of any other item type, as a handle. This is useful when the control text is too large to fit in a Pascal string returned by [CWPanelGetItemText](#).

The returned handle becomes the property of the caller, and should be disposed when no longer needed. Alternatively, a plug-in may modify the handle's text content, and pass it back to the IDE using [CWPanelSetItemTextHandle](#) to set the item's text. In this case,

since [CWPanelSetItemTextHandle](#) assumes ownership of the handle passed to it, the handle need not be disposed.

The returned handle should be locked prior to access using [CWLockMemHandle](#), and unlocked using [CWUnlockMemHandle](#). The size of the handle may be determined by calling [CWGetMemHandleSize](#), and changed by calling [CWResizeMemHandle](#).

See Also  
[“CWPanelSetItemTextHandle” on page 443](#)  
[“CWPanelGetItemText” on page 429](#)  
[“CWPanelGetValue” on page 431](#)

## CWPanelGetValue

Description	Returns the value of a panel control as a long integer.
Prototype	#include <CWDropInPanel.h>  CW_CALLBACK CWPanelGetValue( <a href="#">CWPluginContext</a> context, long whichItem, long* value);
Parameters	Parameters for this function are:  context              Private, opaque IDE state. whichItem            Specifies the ID of the item for which to return a value. value                Returns the value of the specified control as a long integer. For text controls, this value is obtained by attempting to convert the item text to a number.
Remarks	CWPanelGetValue returns the value of a panel control. CWPanelGetValue returns the following values for common control types:

Control type:	Values returned:
checkbox	0 when clear; 1 when set.
radio button	0 when clear; 1 when set.
popup menu	A number from 1 to the number of items in the popup, indicating the item currently selected.
static text	The value of the text converted to a number.
edit text	The value of the text converted to a number.

If the dialog item is a text field, CWPanelGetValue assumes the text is a decimal number and attempts to convert it to a numeric value. This is useful when validating numeric fields. If the text of the specified field cannot be converted to a long integer, CWPanelGetValue returns an error code.

**See Also**

[“CWPanelGetItemText” on page 429](#)

[“CWPanelSetItemValue” on page 444](#)

[“reqItemHit Request” on page 365](#)

[“Validating Input” on page 373](#)

## CWPanelGetListSelectedItem

Description Get the text of a Windows combo box list item.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelGetListSelectedItem(
    CWPluginContext context,
    long dlgItemID,
    long index,
    char* str,
    short maxLen);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

dlgItemID Specifies the ID of a combo box.

	<b>index</b>	Specifies the one-based index of the combo box list item to return.
	<b>str</b>	Provides space in which the IDE returns the text of the specified list item as a C string.
	<b>maxLen</b>	Specifies the maximum number of characters to return in <b>str</b> , including null byte.
<b>Remarks</b>	Panels use <code>CWPanelGetListItemText</code> to obtain the text of a Windows combo box list item. This is useful for determining the selected item in a combo box list constructed at run time (for example, a list of fonts).	
	<code>CWPanelGetListItemText</code> returns at most <code>maxLen</code> characters of the text. If the item is longer than <code>maxLen</code> , no error is returned, and <code>CWPanelGetListItemText</code> returns as many characters as will fit in <code>str</code> .	
<b>See Also</b>	<a href="#">“CWPanelSetListItemText” on page 445</a> <a href="#">“CWPanelInsertListItem” on page 438</a> <a href="#">“CWPanelDeleteListItem” on page 422</a>	

## **CWPanelGetNumBaseDialogItems**

<b>Description</b>	Returns the number of controls in the CodeWarrior <b>Target Settings</b> dialog prior to installation of a panel's controls.
<b>Prototype</b>	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelGetNumBaseDialogItems(     <a href="#">CWPluginContext</a> context,     short*           baseItems);</pre>
<b>Parameters</b>	Parameters for this function are:
	<b>context</b> Private, opaque IDE state. <b>baseItems</b> Returns the original number of items in the <b>Target Settings</b> dialog, prior to installation of a panel's items.
<b>Remarks</b>	Plug-ins call <code>CWPanelGetNumBaseDialogItems</code> when responding to a <a href="#">reqItemHit</a> request. The value returned is used to

adjust the item number returned by [CWPanelGetDialogItemHit](#). To determine the true item number of the hit item, plug-ins should subtract the value returned by [CWPanelGetNumBaseDialogItems](#) from the item number returned by [CWPanelGetDialogItemHit](#).

See Also [“CWPanelGetDialogItemHit” on page 425](#)

[“reqItemHit” on page 510](#)

[“Determining the “Hit” Dialog Item” on page 350](#)

## **CWPanelGetOriginalPrefs**

Description Returns a handle containing the panel’s original settings data.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelGetOriginalPrefs(
    CWPluginContext context,
    CWMemHandle* originalPrefs);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

originalPrefs Returns a handle containing the plug-in’s most recently committed settings data.

Remarks A plug-in calls [CWPanelGetFactoryPrefs](#) to obtain the IDE’s most recently saved copy of the panel’s settings data. The handle returned remains the IDE’s property. plug-ins should not modify the contents of this handle.

The handle returned contains the settings data values most recently saved by the user. If the user has not made any changes in the **Target Settings** dialog, the contents of the original and current settings will be identical.

Before accessing the returned settings handle, a plug-in should call [CWLockMemHandle](#) to lock the handle. When finished accessing the handle, a plug-in should call [CWUnlockMemHandle](#). To determine the size of the handle, use [CWGetMemHandleSize](#). To resize the handle, use [CWResizeMemHandle](#).

Mac OS      On Mac OS, plug-ins should instead directly access the `factoryPrefs` field of the [PanelParameterBlock](#). The handle should be locked, unlocked, and resized using the Mac OS toolbox calls `HLock`, `HUnlock`, and `GetHandleSize` and `SetHandleSize`.

See Also     [“CWPanelGetCurrentPrefs” on page 423](#)

[“CWPanelGetFactoryPrefs” on page 426](#)

[“CWPanelGetPanelPrefs” on page 435](#)

[“Settings Panel Data” on page 342](#)

[“reqItemHit Request” on page 365](#)

[“CWPanelSetFactoryFlag” on page 440](#)

[“CWPanelSetRevertFlag” on page 449](#)

## CWPanelGetPanelPrefs

Description    Returns a handle containing settings data for another panel, specified by name.

Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelGetPanelPrefs(
    CWPluginContext    context,
    const char*           panelName,
    CWMemHandle*      prefs,
    Boolean*              requiresByteSwap);
```

Parameters   Parameters for this function are:

context        Private, opaque IDE state.

panelName      Specifies the name of the panel whose settings data handle should be returned. Specified as a C string on Windows, and as a Pascal string on Mac OS.

prefs           Returns a handle containing the named panel's settings data.

requiresByteSwap    Returns true if the data in the `prefs` handle must be swapped prior to use.

## Settings Panel Plug-in API Reference

### `CWPanelGetRelativePathString`

---

Remarks	<p>Plug-ins call <code>CWPanelGetPanelPrefs</code> to obtain the settings data of a different panel. This can be useful when multiple panels are required to set up a compiler, linker, or version control system's data, or when one panel constructs some of its data values based on setting configured in another panel.</p> <p>A plug-in must know the format of another panel's data returned by <code>CWPanelGetPanelPrefs</code> to utilize it effectively. Normally, only closely related settings panels should share data this way.</p> <p>If <code>requiresByteSwap</code> is true on return, the byte ordering of the data must be swapped before use.</p> <p>Plug-ins should not modify the contents of the data handle returned. The handle remains the property of the IDE, and should <i>not</i> be disposed by the plug-in.</p> <p>Before accessing the returned settings handle, a plug-in should call <a href="#"><code>CWLockMemHandle</code></a> to lock the handle. When finished accessing the handle, a plug-in should call <a href="#"><code>CWUnlockMemHandle</code></a>. To determine the size of the handle, use <a href="#"><code>CWGetMemHandleSize</code></a>. To resize the handle, use <a href="#"><code>CWResizeMemHandle</code></a>.</p>
---------	--

#### See Also

[“CWPanelGetCurrentPrefs” on page 423](#)

[“CWPanelGetFactoryPrefs” on page 426](#)

[“CWPanelGetOriginalPrefs” on page 434](#)

[“Settings Panel Data” on page 342](#)

[“reqItemHit Request” on page 365](#)

[“CWPanelSetFactoryFlag” on page 440](#)

[“CWPanelSetRevertFlag” on page 449](#)

## **CWPanelGetRelativePathString**

Description Converts a relative path to a textual full path representation.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelGetRelativePathString(
    CWPluginContext context,
```

```
CWRelativePath* inPath,
char* pathString,
long* maxLength);
```

Parameters	Parameters for this function are:
context	Private, opaque IDE state.
inPath	Specifies the relative path to convert to text.
pathString	Returns the full path equivalent of the relative path specified by inPath, as a C string on Windows, and as a Pascal string on Mac OS. The returned string includes special path prefixes, such as '{Project}' to indicate the reference location for relative paths.
maxLength	Specifies the maximum length of the text to return in pathString, including length NULL byte.

Remarks CWPanelGetRelativePathString returns a path string for a given relative path in a form suitable for display to users. For example, the IDE uses CWPanelGetRelativePathString to display the **Output Directory** path in the **Target Settings** dialog.

[CWRelativePath](#) structures consist of a reference directory and a partial path. The partial path specifies the location of an item relative to the reference directory. Path strings returned by CWPanelGetRelativePathString start with prefixes which indicate the type of reference path. ([CWResolveRelativePath](#), in contrast, always returns the full path.)

Depending upon the value of pathType for a [CWRelativePath](#) structure, CWPanelGetRelativePathString constructs path strings beginning with one of the following prefixes:

<b>Path type:</b>	<b>Path string prefix:</b>
type_Compiler	'{Compiler}'
type_Project	'{Project}'

<b>Path type:</b>	<b>Path string prefix:</b>
type_System	'{System}'
type_Absolute,	None; output is full path.
type_UserDefined	

For example, if the output directory is set to a 'bin' subdirectory of the current project, CWPanellInsertListltem will convert the relative path for the output directory to the following string:

'{Project}\bin'

Panels may call [CWPanellChooseRelativePath](#) to request selection of a relative path by the user. [CWRelativePath](#) structures may also be constructed explicitly.

See Also [“CWPanellChooseRelativePath” on page 420](#)

[“CWRelativePath” on page 178](#)

[“CWRelativePathTypes” on page 181](#)

[“CWResolveRelativePath” on page 142](#)

## **CWPanellInsertListltem**

Description Inserts an item into a Windows combo box list.

Prototype

```
#include <CWDropInPanel.h>
CWPanellInsertListltem(
    CWPluginContext context,
    long dlgItemID,
    long index,
    char* str);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

dlgItemID Specifies the ID of the combo box.

	<b>index</b>	Specifies the position at which to insert the text item in the combo box list. Item indexes are one-based.
	<b>str</b>	Specifies the text of the item to insert, as a C string.
<b>Remarks</b>	Use <code>CWPanellInsertListItem</code> to insert items in a Windows combo box list. plug-ins typically call <code>CWPanellInsertListItem</code> in response to a <a href="#">reqInitDialog</a> request, to initialize the contents of a combo box list.	
	Plug-ins can also call <code>CWPanellInsertListItem</code> while a panel is running, to modify the options presented in a combo box (such as a history list). To delete items from a combo box list, call <a href="#">CWPanellDeleteListItem</a> .	
	Items are inserted at the position specified by <code>index</code> , and all other items initially at the same index or higher are shuffled to higher indexes. For example, inserting “Foo” at position 2 in the list {“First Item”, “Last Item”} yields the list {“First Item”, “Foo”, “Last Item”}.	
	To insert items at the start of the list, use a value of 1 for <code>index</code> . To insert items at the end of the list, specify an <code>index</code> value one higher than the number of items in the list.	
<b>See Also</b>	<a href="#">“CWPanellDeleteListItem” on page 422</a> <a href="#">“reqInitDialog” on page 509</a> <a href="#">“CWPanellGetListItemText” on page 432</a> <a href="#">“CWPanellSetListItemText” on page 445</a>	

## **CWPanellInvalItem**

**Description** Invalidates the dialog area occupied by a dialog item.

**Prototype**

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanellInvalItem(
    CWPluginContext context,
    long whichItem);
```

**Parameters** Parameters for this function are:

	<b>context</b>	Private, opaque IDE state.
	<b>whichItem</b>	Specifies the ID of the panel item to invalidate.
<b>Remarks</b>	Plug-ins call <code>CWPanelInvalItem</code> to force an item to be redrawn. The item will be redrawn the next time the IDE receives a paint message. This is most commonly used for custom items. Most plug-ins do not need to call this routine.	
<b>See Also</b>	<a href="#">“CWPanelValidItem” on page 451</a> <a href="#">“Redrawing Panel Items” on page 374</a>	

## **CWPanelSetFactoryFlag**

<b>Description</b>	Sets the value of the factory settings flag.
<b>Prototype</b>	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelSetFactoryFlag(     <a href="#">CWPluginContext</a> context,     Boolean canFactory);</pre>
<b>Parameters</b>	Parameters for this function are:
	<b>context</b> Private, opaque IDE state.
	<b>canFactory</b> Specifies the new value of the factory flag. Set this to true if the current preferences differ from the factory preferences.
<b>Remarks</b>	Plug-ins call <code>CWPanelSetFactoryFlag</code> to set the state of the factory settings flag. This flag indicates whether the current settings matches the factory settings. A true value means the current settings do not match factory settings. When true, the IDE will enable the <b>Factory Settings</b> button in the <b>Target Settings</b> dialog, allowing the user to reset all settings to their factory defaults.  This flag should be set or cleared in response to <a href="#">reqItemHit</a> request. To obtain the plug-in's current settings data, call <a href="#">CWPanelGetCurrentPrefs</a> . To obtain its current factory settings, call <a href="#">CWPanelGetFactoryPrefs</a> .
<b>See Also</b>	<a href="#">“reqItemHit” on page 510</a> <a href="#">“reqItemHit Request” on page 365</a>

- [“CWPanelSetRevertFlag” on page 449](#)
- [“CWPanelGetCurrentPrefs” on page 423](#)
- [“CWPanelGetFactoryPrefs” on page 426](#)
- [“CWPanelGetOriginalPrefs” on page 434](#)

## **CWPanelSetItemData**

Description	Sets additional type-specific data for a panel control.								
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelSetItemData(     <a href="#">CWPluginContext</a>    context,     long                  dlgItemID,     void*                inData,     long                  inDataLength);</pre>								
Parameters	Parameters for this function are:								
	<table border="0"><tr><td>context</td><td>Private, opaque IDE state.</td></tr><tr><td>dlgItemID</td><td>Specifies the panel item for which to obtain additional information.</td></tr><tr><td>inData</td><td>Points to data of the appropriate type for the item.</td></tr><tr><td>inDataLength</td><td>Specifies the length of the data pointed to be inData.</td></tr></table>	context	Private, opaque IDE state.	dlgItemID	Specifies the panel item for which to obtain additional information.	inData	Points to data of the appropriate type for the item.	inDataLength	Specifies the length of the data pointed to be inData.
context	Private, opaque IDE state.								
dlgItemID	Specifies the panel item for which to obtain additional information.								
inData	Points to data of the appropriate type for the item.								
inDataLength	Specifies the length of the data pointed to be inData.								
Remarks	This routine provides no useful service for third-party plug-in developers at this time. The exact behavior of this routine is subject to change. Plug-ins should not use this routine.								
See Also	<a href="#">“CWPanelGetItemData” on page 427</a>								

## **CWPanelSetItemMaxLength**

Description	Sets the maximum text entry length for an edit text control.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelSetItemMaxLength(     <a href="#">CWPluginContext</a>    context,     long                  dlgItemID,</pre>

```
short           inLength);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	dlgItemID    Specifies the ID of the item for which to set the maximum text length.
	inLength     Specifies the maximum allowed text length, in bytes, including the length/NUL byte.
Remarks	Call <code>CWPanelSetItemMaxLength</code> to set the maximum number of characters the user may enter into an edit text control.  A plug-in typically calls <code>CWPanelSetItemMaxLength</code> when initializing its UI in response to a <a href="#">reqInitDialog</a> request, and occasionally when responding to a <a href="#">reqItemHit</a> request.
See Also	<a href="#">“CWPanelGetItemMaxLength” on page 428</a> <a href="#">“CWPanelGetItemText” on page 429</a> <a href="#">“CWPanelSetItemText” on page 442</a> <a href="#">“Limiting text input” on page 372</a> <a href="#">“Validating Input” on page 373</a>

## **CWPanelSetItemText**

Description	Sets the text of a dialog item.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelSetItemText(     <a href="#">CWPluginContext</a> context,     long             whichItem,     char*           str);</pre>
Parameters	Parameters for this function are:

context      Private, opaque IDE state.

whichItem	Specifies the ID of the item for which to set the text.
str	Specifies the new text for the control, as a C string on Windows, and as a Pascal string on Mac OS.
Remarks	This routine changes the text for an editable text field, static text, or an item's title.  A plug-in typically uses <code>CWPanelSetText</code> to install text in controls in response to a <a href="#">reqPutData</a> request. Plug-ins may also call <code>CWPanelSetText</code> to modify the text of an item in response to a <a href="#">reqItemHit</a> request.  For example, if the user enters invalid text in a field, the plug-in can extract the text from a field using <a href="#">CWPanelGetItemText</a> , remove the invalid characters, and place the cleaned up text in the item using <code>CWPanelSetText</code> .
See Also	<a href="#">“CWPanelGetItemText” on page 429</a> <a href="#">“CWPanelGetItemValue” on page 431</a> <a href="#">“CWPanelSetValue” on page 444</a> <a href="#">“reqPutData Request” on page 355</a> <a href="#">“reqItemHit Request” on page 365</a> <a href="#">“reqPutData” on page 512</a> <a href="#">“reqItemHit” on page 510</a>

## **CWPanelSetItemTextHandle**

Description	Sets the text or title of a panel control to the text contained in a handle.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanelSetItemTextHandle(     <a href="#">CWPluginContext</a> context,     long whichItem,     <a href="#">CWMemHandle</a> text);</pre>

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	whichItem    Specifies the ID of the item for which to set the text.
	text          Contains the new text of the item.
Remarks	Plug-ins call <code>CWPanelSetItemTextHandle</code> to change the text of a static text or edit text item, or the title of any other control type.  Unlike <a href="#">CWPanelSetItemText</a> , which only provides access to the first 255 characters of an item's text (on Mac OS), <code>CWPanelSetItemTextHandle</code> allows a plug-in to set the entire text of an item.  The text handle becomes the property of the IDE. Call <a href="#">CWPanelGetItemTextHandle</a> to get the text of an item in a handle. A convenient way to make changes to an item's text, is to get the item's text by calling <a href="#">CWPanelGetItemTextHandle</a> , make changes to the text, and then call <code>CWPanelSetItemTextHandle</code> to set it.
See Also	<a href="#">“CWPanelGetItemTextHandle” on page 430</a> <a href="#">“CWPanelGetItemText” on page 429</a> <a href="#">“CWPanelSetItemText” on page 442</a> <a href="#">“CWPanelGetValue” on page 431</a> <a href="#">“CWPanelSetValue” on page 444</a>

## **CWPanelSetValue**

Description	Sets the text of an static or edit text item to the text representation of a long integer.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanelSetValue( <a href="#">CWPluginContext</a> context, long                  whichItem, long                  value);
Parameters	Parameters for this function are:

context	Private, opaque IDE state.
whichItem	Sets the ID of the item for which to change the value.
value	Specifies the new value of the text item.

**Remarks** `CWPanelSetValue` sets the value of a panel control. `CWPanelSetValue` accepts the following values for common control types:

<b>Control type:</b>	<b>Values to install in control:</b>
checkbox	0 to clear; 1 to set.
radio button	0 to clear; 1 to set.
popup menu	A number from 1 to the number of items in the popup, indicating the item currently selected.
static text	The value of the text converted to a number.
edit text	The value of the text converted to a number.

If the dialog item is a text field, `CWPanelSetValue` sets the item to the text representation of a long integer. This is convenient when dealing with numeric entry fields.

For example, when responding to a `reqItemHit` request, a panel can obtain the value of a text field interpreted as a number by calling `CWPanelGetValue`. If the value returned is out of range, the plug-in can install a properly constrained value in the text field using `CWPanelSetValue`.

**See Also**

[“CWPanelGetValue” on page 431](#)  
[“CWPanelGetItemText” on page 429](#)  
[“CWPanelSetText” on page 442](#)

## **CWPanelSetListltemText**

**Description** Sets the text of a Windows combo box list item.  
**Prototype** `#include <CWDropInPanel.h>`

---

## Settings Panel Plug-in API Reference

### CWPanelSetRecompileFlag

---

```
CW_CALLBACK CWPanelSetListListItemText(
    CWPluginContext context,
    long dlgItemID,
    long index,
    char* str);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	dlgItemID    Specifies the ID of the combo box item.
	index        Specifies the one-based index of the combo box item for which to set the text content.
	str          Specifies the new text of the combo box list item, as a C string.
Remarks	A panel calls <code>CWPanelSetListListItemText</code> to set the text of a Windows combo box list item. Combo box list indexes are one-based. To obtain the text of an item, call <a href="#">CWPanelGetListListItemText</a> .  Plug-ins call <code>CWPanelSetListListItemText</code> to install items in a combo box during dialog initialization, in response to a <a href="#">reqInitDialog</a> request. Plug-ins may also call <code>CWPanelSetListListItemText</code> to change the text of combo box list items in response to user actions (typically during a <a href="#">reqItemHit</a> request).
See Also	<a href="#">“CWPanelGetListListItemText” on page 432</a> <a href="#">“CWPanelInsertListItem” on page 438</a> <a href="#">“CWPanelDeleteListItem” on page 422</a> <a href="#">“reqInitDialog” on page 509</a> <a href="#">“reqItemHit” on page 510</a>

## CWPanelSetRecompileFlag

Description	Sets the value of the recompile flag.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanelSetRecompileFlag(

```
CWPluginContext context,  
Boolean recompile);
```

- Parameters    Parameters for this function are:
- context        Private, opaque IDE state.
- recompile      Specifies the new value of the recompile flag.

Remarks    A plug-in calls `CWPanelSetRecompileFlag` when responding to a [reqValidate](#) request.

A plug-in should set this flag if its current settings data necessitates a recompile of the the current target's source code. This might be necessary, for example, if the user of a panel changes the optimization level of compiled code.

- See Also    [“reqValidate” on page 516](#)  
[“reqValidate Request” on page 361](#)  
[“CWPanelSetRelinkFlag” on page 447](#)  
[“CWPanelSetReparseFlag” on page 448](#)  
[“CWPanelSetResetPathsFlag” on page 449](#)

## **CWPanelSetRelinkFlag**

- Description    Sets the value of the relink flag.
- Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelSetRelinkFlag(
    CWPluginContext context,
    Boolean relink);
```
- Parameters    Parameters for this function are:
- context        Private, opaque IDE state.
- relink         Specifies the new value of the relink flag.
- Remarks    A plug-in calls `CWPanelSetRelinkFlag` when responding to a [reqValidate](#) request.

A plug-in should set this flag if its current settings data necessitates a relink of the the current target's compiled executable. This might be necessary, for example, if the user of a panel changes the executable binary type of the current target.

**See Also**

- [“reqValidate” on page 516](#)
- [“reqValidate Request” on page 361](#)
- [“CWPanelSetRecompileFlag” on page 446](#)
- [“CWPanelSetReparseFlag” on page 448](#)
- [“CWPanelSetResetPathsFlag” on page 449](#)

## **CWPanelSetReparseFlag**

**Description** Sets the value of the reparse flag.

**Prototype**

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelSetReparseFlag(
    CWPluginContext context,
    Boolean           reparse);
```

**Parameters** Parameters for this function are:

- context** Private, opaque IDE state.
- reparse** Specifies the new value of the reparse flag.

**Remarks** A plug-in calls `CWPanelSetReparseFlag` when responding to a [reqValidate](#) request. A panel should set this flag if changes to its settings affect the information maintained by the IDE's symbol browser.

---

**NOTE** This flag is not currently used by the IDE. Most plug-ins do not need to set this flag.

---

**See Also**

- [“reqValidate” on page 516](#)
- [“reqValidate Request” on page 361](#)
- [“CWPanelSetRecompileFlag” on page 446](#)

[“CWPanelSetRelinkFlag” on page 447](#)

[“CWPanelSetResetPathsFlag” on page 449](#)

## **CWPanelSetResetPathsFlag**

Description Sets the value of the “reset paths” settings flag.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelSetResetPathsFlag(
    CWPluginContext context,
    Boolean           resetPaths);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

resetPaths Specifies the new value of the “reset paths” flag.

Remarks A plug-in calls `CWPanelSetResetPathsFlag` when responding to a [reqValidate](#) request.

A plug-in should set this flag if its current settings data requires the IDE to rescan for project files. When a plug-in sets this flag, the IDE will search for all target files within the currently established access paths, after the user commits (saves) the changes made in the panel. This might be necessary, for example, if the user of a panel adds or removes project paths.

See Also [“reqValidate” on page 516](#)

[“reqValidate Request” on page 361](#)

[“CWPanelSetRecompileFlag” on page 446](#)

[“CWPanelSetRelinkFlag” on page 447](#)

[“CWPanelSetReparseFlag” on page 448](#)

## **CWPanelSetRevertFlag**

Description Sets the value of the revert flag.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelSetRevertFlag(
```

```
CWPluginContext context,  
Boolean canRevert);
```

Parameters    Parameters for this function are:

context        Private, opaque IDE state.

canRevert      Specifies the new value of the revert flag.

Remarks    A plug-in calls `CWPanelSetRevertFlag` when responding to a [reqItemHit](#) request.

A plug-in should set this flag to indicate that the current settings data differs from the most recently saved settings data. When set, the IDE enables the **Revert** button in the **Target Settings** dialog.

See Also    [“CWPanelSetFactoryFlag” on page 440](#)

[“reqItemHit” on page 510](#)

[“reqItemHit Request” on page 365](#)

## **CWPanelShowItem**

Description    Shows or hides an item in a settings panel.

```
#include <CWDropInPanel.h>  
CW_CALLBACK CWPanelShowItem(  
    CWPluginContext context,  
    long whichItem,  
    Boolean showIt);
```

Parameters    Parameters for this function are:

context        Private, opaque IDE state.

whichItem      Specifies the ID of the item to show or hide.

showIt         Specifies whether to show the item (nonzero) or  
                hide it (zero).

Remarks    A plug-in calls this routine to show or hide a dialog item.

`CWPanelShowItem` may be useful for hiding controls which, based upon other target settings, have no relevance. Usually, however, controls should not be hidden from the user; in most cases, enabling

and disabling items using [CWPanelEnableItem](#) is more appropriate.

**TIP** In general, if an item can ever be enabled during use of its panel, it should not be hidden. Tabbed panels are a good exception to this rule.

---

When showing controls that were previously hidden, you may want to activate one of them using [CWPanelEnableItem](#), to give it input focus.

See Also [“CWPanelEnableItem” on page 423](#)

[“CWPanelActivateItem” on page 418](#)

[“Enabling and Disabling Items” on page 372](#)

## **CWPanelValidItem**

Description Validates a panel control.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanelValidItem(
    CWPluginContext context,
    long whichItem);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

whichItem Specifies the ID of the item to validate.

Remarks Call `CWPanelValidItem` to inform the IDE and host operating system that a control need not be redrawn. `CWPanelValidItem` cancels any pending paint (update) messages for the specified item.

Most panels do not need to call `CWPanelValidItem`. `CWPanelValidItem` is most useful for custom controls.

See Also [“CWPanelInvalItem” on page 439](#)

[“Redrawing Panel Items” on page 374](#)

## **CWPanlActivateItem**

Description      The Mac OS version of [CWPanlActivateItem](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlActivateItem(
    PanelParamBlkPtr ppb,
    long                 whichItem);
```

## **CWPanlAppendItems**

Description      The Mac OS version of [CWPanlAppendItems](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlAppendItems(
    PanelParamBlkPtr ppb,
    short                ditlID);
```

## **CWPanlChooseRelativePath**

Description      The Mac OS version of [CWPanlChooseRelativePath](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlChooseRelativePath(
    PanelParamBlkPtr ppb,
    CWRelativePath\* path,
    Boolean              isFolder,
    short                filterCount,
    void*                filterList,
    char*                prompt);
```

## **CWPanlDrawPanelBox**

Description      Draws a rectangle around a dialog item and optionally draws a title for the rectangle.

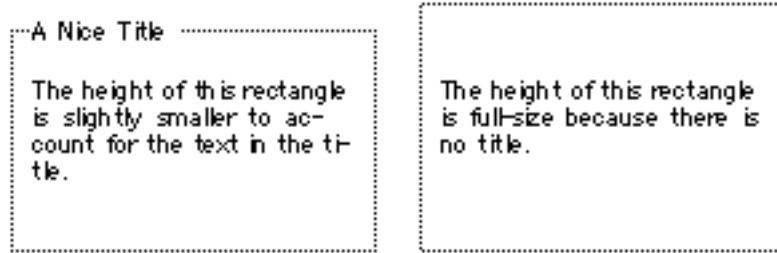
Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlDrawPanelBox(
    PanelParamBlkPtr ppb,
    long                  whichItem,
    ConstStr255Param     title);
```

Parameters     Parameters for this function are:

ppb	Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.
whichItem	The plug-in specifies the dialog item to draw the box around.
title	The plug-in specifies the title of the box, formatted as a Pascal string.
Remarks	<p>Draws a rectangle around a dialog box item and optionally draws a title for the rectangle.</p> <p>The IDE draws a rectangle around the dialog item specified by <i>whichItem</i>. <i>title</i> specifies the title to give the rectangle. If <i>title</i> is an empty string, no title is drawn (<a href="#">Figure 11.1</a>).</p>

**Figure 11.1 How CWPanlDrawPanelBox draws a rectangle**



See Also      [“Drawing Custom Items” on page 392](#)

## CWPanlDrawUserItemBox

Description	Obsolete call to the IDE.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanlDrawUserItemBox(     DialogPtr           dialog,     short                whichItem,     ConstStr255Param   title);</pre>

## CWPanlEnableItem

Description	The Mac OS version of <a href="#">CWPanlEnableItem</a> .
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanlEnableItem(</pre>

```
PanelParamBlkPtr ppb,  
long whichItem,  
Boolean enableIt);
```

## **CWPanlGetArraySettingElement**

Description The Mac OS version of [CWGetArraySettingElement](#).

Prototype

```
#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlGetArraySettingElement(  
    PanelParamBlkPtr ppb,  
    CWSettingID settingID,  
    long index,  
    CWSettingID* elementSettingID);
```

## **CWPanlGetArraySettingSize**

Description The Mac OS version of [CWGetArraySettingSize](#).

Prototype

```
#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlGetArraySettingSize(  
    PanelParamBlkPtr ppb,  
    CWSettingID settingID,  
    long* size);
```

## **CWPanlGetBooleanValue**

Description The Mac OS version of [CWGetBooleanValue](#).

Prototype

```
#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlGetBooleanValue(  
    PanelParamBlkPtr ppb,  
    CWSettingID settingID,  
    Boolean* value);
```

## **CWPanlGetFloatingPointValue**

Description The Mac OS version of [CWGetFloatingPointValue](#).

Prototype

```
#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlGetFloatingPointValue(  
    PanelParamBlkPtr ppb,  
    CWSettingID settingID,
```

```
double* value);
```

## **CWPanlGetIntegerValue**

Description The Mac OS version of [CWGetIntegerValue](#).

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetIntegerValue(
    PanelParamBlkPtr ppb,
    CWSettingID settingID,
    long* value);
```

## **CWPanlGetItemControl**

Description Retrieves the Mac OS control handle for a dialog item.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetItemControl(
    PanelParamBlkPtr ppb,
    long whichItem,
    ControlRef* control);
```

Parameters Parameters for this function are:

ppb	Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.
whichItem	Specifies the index of the item for which to return a control.
control	Returns the Mac OS control for the specified panel item.

Remarks A plug-in calls this routine to get the control handle for the item specified by whichItem. For items that contain menus, a menu handle will be returned instead.

In rare cases, a plug-in may require access to the Mac OS control corresponding to a panel item. To obtain this control, call CWPanlGetItemControl with the index of the panel item.

---

**NOTE** Whenever possible, plug-ins should use API routines to access controls, rather than Mac OS toolbox calls.

---

The returned control handle remains property of the IDE. If an item has no associated control, CWPanlGetItemControl will return an error.

See Also [“CWPanlGetMacPort” on page 458](#)

## CWPanlGetItemData

Description The Mac OS version of [CWPanlGetItemData](#).

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetItemData(
    PanelParamBlkPtr ppb,
    long                  whichItem,
    void*                outData,
    long*                outDataLength);
```

## CWPanlGetItemMaxLength

Description The Mac OS version of [CWPanlGetItemMaxLength](#).

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetItemMaxLength(
    PanelParamBlkPtr ppb,
    long                  whichItem,
    short*               outLength);
```

## CWPanlGetItemRect

Description Returns the bounding rectangle of the dialog item.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetItemRect(
    PanelParamBlkPtr ppb,
    long                  whichItem,
    Rect*                rect);
```

Parameters Parameters for this function are:

ppb Parameter block passed to Mac OS panels,  
required by the IDE to service plug-in requests.

	whichItem	Specifies the index of the item for which to get the bounding rectangle.
	rect	The IDE returns the rectangle in this argument.
Remarks	A plug-in calls this routine to get the bounding rectangle of a dialog item, in local coordinates. This is most useful when drawing and hit-testing custom controls (user items).	
See Also	<a href="#">“CWPanlGetItemControl” on page 455</a> <a href="#">“Drawing Custom Items” on page 392</a>	

## **CWPanlGetItemText**

Description	The Mac OS version of <a href="#">CWPanlGetItemText</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlGetItemText( <a href="#">PanelParamBlkPtr</a> ppb, long                       whichItem, StringPtr                 str, short                     maxLen);

## **CWPanlGetItemTextHandle**

Description	The Mac OS version of <a href="#">CWPanlGetItemTextHandle</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlGetItemTextHandle( <a href="#">PanelParamBlkPtr</a> ppb, long                       whichItem, Handle*                 text);

## **CWPanlGetValue**

Description	The Mac OS version of <a href="#">CWPanlGetValue</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlGetValue( <a href="#">PanelParamBlkPtr</a> ppb, long                       whichItem, long*                     value);

# CWPanIGetMacPort

Description	Returns the graphics port of the settings dialog window.				
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; CW_CALLBACK CWPanlGetMacPort(     <u>PanelParamBlkPtr</u> ppb,     GrafPtr*           port);</pre>				
Parameters	Parameters for this function are:				
	<table><tr><td>ppb</td><td>Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.</td></tr><tr><td>port</td><td>Returns the QuickDraw graphics port of the <b>Target Settings</b> dialog box.</td></tr></table>	ppb	Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.	port	Returns the QuickDraw graphics port of the <b>Target Settings</b> dialog box.
ppb	Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.				
port	Returns the QuickDraw graphics port of the <b>Target Settings</b> dialog box.				
Remarks	CWPanlGetMacPort returns a pointer to the Mac OS graphics port in which all settings dialog drawing takes place.  This is most useful when implementing custom controls. For example, a user item may wish to change the pen size, foreground or background colors, text font or size, or the clipping region.				
<b>WARNING!</b>	A panel should always restore any graphics port characteristics it has changed before returning to the IDE.				
See Also	<a href="#">“CWPanlDrawUserItemBox” on page 453</a> <a href="#">“Drawing Custom Items” on page 392</a>				

## **CWPanIGetNamedSetting**

Description	The Mac OS version of <a href="#">CWGetNamedSetting</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlGetNamedSetting( <u>PanelParamBlkPtr</u> ppb, const char* name, CWSettingID* settingID);

## CWPan|GetPanelPrefs

Description The Mac OS version of [CWPPanelGetPanelPrefs](#).

Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetPanelPrefs(
    PanelParamBlkPtr ppb,
    StringPtr           inPanelName,
    Handle*             prefs,
    Boolean*            requiresByteSwap);
```

## **CWPanlGetRelativePathString**

Description    The Mac OS version of [CWPanlGetRelativePathString](#).

Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetRelativePathString(
    PanelParamBlkPtr ppb,
    CWRelativePath\* path,
    char*               pathString,
    long*               maxLength);
```

## **CWPanlGetRelativePathValue**

Description    The Mac OS version of [CWGetRelativePathValue](#).

Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetRelativePathValue(
    PanelParamBlkPtr ppb,
    CWSettingID      settingID,
    CWRelativePath\* value);
```

## **CWPanlGetStringValue**

Description    The Mac OS version of [CWGetStringValue](#).

Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlGetStringValue(
    PanelParamBlkPtr ppb,
    CWSettingID      settingID,
    const char**        value);
```

## **CWPanlGetStructureSettingField**

Description    The Mac OS version of [CWGetStructureSettingField](#).

Prototype    

```
#include <CWDropInPanel.h>
```

```
CW_CALLBACK CWPanlGetStructureSettingField(
    PanelParamBlkPtr ppb,
    CWSettingID      settingID,
    const char*         name,
    CWSettingID*     fieldSettingID);
```

## **CWPanlInstallUserItem**

Description      Obsolete call to the IDE.

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlInstallUserItem(
    PanelParamBlkPtr ppb,
    short             whichItem,
    UserItemProcPtr   proc);
```

## **CWPanlInvalItem**

Description      The Mac OS version of [CWPanlInvalItem](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlInvalItem(
    PanelParamBlkPtr ppb,
    long              whichItem);
```

## **CWPanlReadBooleanSetting**

Description      The Mac OS version of [CWReadBooleanSetting](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlReadBooleanSetting(
    PanelParamBlkPtr ppb,
    const char*       name,
    Boolean*          value);
```

## **CWPanlReadFloatingPointSetting**

Description      The Mac OS version of [CWReadFloatingPointSetting](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlReadFloatingPointSetting(
    PanelParamBlkPtr ppb,
    const char*       name,
```

```
double* value);
```

## **CWPanlReadIntegerSetting**

Description The Mac OS version of [CWReadIntegerSetting](#).

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlReadIntegerSetting(
    PanelParamBlkPtr ppb,
    const char* name,
    long* value);
```

## **CWPanlReadRelativePathAEDesc**

Description Extracts a relative path from a given Apple Event descriptor record.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlReadRelativePathAEDesc (
    PanelParamBlkPtr ppb,
    CWRelativePath\* path,
    const AEDesc* desc);
```

Parameters Parameters for this function are:

- |      |   |
|------|---|
| ppb  | Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.     |
| path | Returns the relative path specified by the Apple Event descriptor passed in desc.             |
| desc | Provides the Apple Event descriptor data from an Apple Event, which contains a relative path. |

Remarks CWPanlReadRelativePathAEDesc simplifies the task of extracting a [CWRelativePath](#) record from an Apple Event descriptor. Plug-ins call CWPanlReadRelativePathAEDesc in response to a [reqAESetPref](#) request, to extract a relative path setting from an Apple Event descriptor.

A plug-in passes the AEDesc from the prefsDesc field of the [PanelParameterBlock](#) in the desc field. The IDE extracts the relative path information from the Apple Event descriptor, and stores it in the path structure. desc remains the property of the plug-in.

See Also    [“CWPanlWriteRelativePathAEDesc” on page 466](#)

## CWPanlReadRelativePathSetting

Description	The Mac OS version of <a href="#">CWReadRelativePathSetting</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlReadRelativePathSetting( <a href="#">PanelParamBlkPtr</a> ppb, const char* name, <a href="#">CWRelativePath</a> * value);

## CWPanlReadStringSetting

Description	The Mac OS version of <a href="#">CWReadStringSetting</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlReadStringSetting( <a href="#">PanelParamBlkPtr</a> ppb, const char* name, const char** value);

## CWPanlRemoveUserItem

Description	Obsolete call to the IDE.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlRemoveUserItem( <a href="#">PanelParamBlkPtr</a> ppb, short whichItem);

## CWPanlSetBooleanValue

Description	The Mac OS version of <a href="#">CWSetBooleanValue</a> .
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWPanlSetBooleanValue( <a href="#">PanelParamBlkPtr</a> ppb, <a href="#">CWSettingID</a> settingID, Boolean value);

## **CWPanlSetFloatingPointValue**

Description      The Mac OS version of [CWSetFloatingPointValue](#).

Prototype     #include <CWDropInPanel.h>  
CW\_CALLBACK CWPanlSetFloatingPointValue(  
    [PanelParamBlkPtr](#) ppb,  
    [CWSettingID](#) settingID,  
    double value);

## **CWPanlSetIntegerValue**

Description      The Mac OS version of [CWSetIntegerValue](#).

Prototype     #include <CWDropInPanel.h>  
CW\_CALLBACK CWPanlSetIntegerValue(  
    [PanelParamBlkPtr](#) ppb,  
    [CWSettingID](#) settingID,  
    long value);

## **CWPanlSetItemData**

Description      The Mac OS version of [CWPanlSetItemData](#).

Prototype     #include <CWDropInPanel.h>  
CW\_CALLBACK CWPanlSetItemData(  
    [PanelParamBlkPtr](#) ppb,  
    long whichItem,  
    void\* inData,  
    long inDataLength);

## **CWPanlSetItemMaxLength**

Description      The Mac OS version of [CWPanlSetItemMaxLength](#).

Prototype     #include <CWDropInPanel.h>  
CW\_CALLBACK CWPanlSetItemMaxLength(  
    [PanelParamBlkPtr](#) ppb,  
    long whichItem,  
    short inLength);

## **CWPanlSetItemText**

Description      The Mac OS version of [CWPanlSetItemText](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlSetItemText(
    PanelParamBlkPtr ppb,
    long                whichItem,
    ConstStr255Param str);
```

## **CWPanlSetItemTextHandle**

Description      The Mac OS version of [CWPanlSetItemTextHandle](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlSetItemTextHandle(
    PanelParamBlkPtr ppb,
    long                whichItem,
    Handle              text);
```

## **CWPanlSetItemValue**

Description      The Mac OS version of [CWPanlSetItemValue](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlSetItemValue(
    PanelParamBlkPtr ppb,
    long                whichItem,
    long                value);
```

## **CWPanlSetRelativePathValue**

Description      The Mac OS version of [CWPanlSetRelativePathValue](#).

Prototype     

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlSetRelativePathValue(
    PanelParamBlkPtr ppb,
    CWSettingID      settingID,
    const CWRelativePath* value);
```

## **CWPanlSetStringValue**

Description      The Mac OS version of [CWPanlSetStringValue](#).

Prototype    `#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlSetValue(  
    PanelParamBlkPtr ppb,  
    CWSettingID settingID,  
    const char* value);`

## CWPanlShowItem

Description    The Mac OS version of [CWPanlShowItem](#).

Prototype    `#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlShowItem(  
    PanelParamBlkPtr ppb,  
    long whichItem,  
    Boolean showIt);`

## CWPanlValidItem

Description    The Mac OS version of [CWPanlValidItem](#).

Prototype    `#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlValidItem(  
    PanelParamBlkPtr ppb,  
    long whichItem);`

## CWPanlWriteBooleanSetting

Description    The Mac OS version of [CWPanlWriteBooleanSetting](#).

Prototype    `#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlWriteBooleanSetting(  
    PanelParamBlkPtr ppb,  
    const char* name,  
    Boolean value);`

## CWPanlWriteFloatingPointSetting

Description    The Mac OS version of [CWPanlWriteFloatingPointSetting](#).

Prototype    `#include <CWDropInPanel.h>  
CW_CALLBACK CWPanlWriteFloatingPointSetting(  
    PanelParamBlkPtr ppb,  
    const char* name,`

```
double value);
```

## CWPanlWriteIntegerSetting

Description The Mac OS version of [CWPanlWriteIntegerSetting](#).

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlWriteIntegerSetting(
    PanelParamBlkPtr ppb,
    const char* name,
    long value);
```

## CWPanlWriteRelativePathAEDesc

Description Constructs an Apple Event descriptor for a given relative path.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlWriteRelativePathAEDesc(
    PanelParamBlkPtr ppb,
    const CWRelativePath* path,
    AEDesc* desc);
```

Parameters Parameters for this function are:

ppb Parameter block passed to Mac OS panels, required by the IDE to service plug-in requests.

path Specifies the relative path for which to construct an Apple Event descriptor.

desc Returns an Apple Event descriptor describing the relative path specified in path. The IDE allocates this descriptor, which becomes the property of the plug-in.

Remarks CWPanlWriteRelativePathAEDesc simplifies the task of creating an Apple Event descriptor for a [CWRelativePath](#) record. Plug-ins call CWPanlWriteRelativePathAEDesc in response to a [reqAEGetPref](#) request, to return an Apple Event descriptor for a relative path setting.

The AEDesc returned in desc becomes the property of the plug-in, and should be disposed when no longer needed, or more typically, returned in the prefsDesc field of the [PanelParameterBlock](#).

See Also [“CWPanlReadRelativePathAEDesc” on page 461](#)

[“reqAEGetPref” on page 499](#)

[“Handling Apple Events” on page 396](#)

## **CWPanlWriteRelativePathSetting**

Description The Mac OS version of [CWPanlWriteRelativePathSetting](#).

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlWriteRelativePathSetting(
    PanelParamBlkPtr          ppb,
    const char*                 name,
    const CWRelativePath*   value);
```

## **CWPanlWriteStringSetting**

Description The Mac OS version of [CWPanlWriteStringSetting](#).

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWPanlWriteStringSetting(
    PanelParamBlkPtr          ppb,
    const char*                 name,
    const char*                 value);
```

## **CWReadBooleanSetting**

Description Reads a boolean value from the top level of a panel’s XML input stream.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWReadBooleanSetting(
    CWPluginContext    context,
    const char*           name,
    Boolean*              value);
```

Parameters Parameters for this function are:

- |         |  |
|---------|--|
| context | Private, opaque IDE state.                                 |
| name    | A C string specifying the name of the XML setting to read. |
| value   | Returns the boolean value of the named setting.            |

**Remarks** CWReadBooleanSetting returns the value of a top-level boolean XML setting (a boolean value contained in the panel's XML stream, but not in a structure or array). If no boolean setting by the specified name is found, an error will be returned.

CWReadBooleanSetting returns an error if the named setting does not exist or is not of boolean type. CWReadBooleanSetting should only be called in response to a [reqReadSettings](#) request.

To read a boolean component of an XML structure or array, use [CWGetBooleanValue](#) instead.

**See Also** [“reqReadSettings” on page 512](#)

[“CWGetBooleanValue” on page 411](#)

## CWReadFloatingPointSetting

**Description** Reads a floating point value from the top level of a panel's XML input stream.

**Prototype**

```
#include <CWDropInPanel.h>
CW_CALLBACK CWReadFloatingPointSetting(
    CWPluginContext context,
    const char*      name,
    double*          value);
```

**Parameters** Parameters for this function are:

**context** Private, opaque IDE state.

**name** A C string specifying the name of the XML setting to read.

**value** Returns the floating point value of the named setting.

**Remarks** CWReadFloatingPointSetting returns the value of a top-level floating point XML setting (a floating point value contained in the panel's XML stream, but not in a structure or array). If no floating point setting by the specified name is found, an error will be returned.

CWReadFloatingPointSetting returns an error if the given setting does not exist or is not of floating point type.

`CWReadFloatingPointSetting` should only be called in response to a [reqReadSettings](#) request.

To read a floating point component of an XML structure or array, use [CWGetFloatingPointValue](#) instead.

See Also [“reqReadSettings” on page 512](#)

[“CWGetFloatingPointValue” on page 411](#)

## **CWReadIntegerSetting**

Description     Reads a long integer value from the top level of a panel's XML input stream.

Prototype    

```
#include <CWDropInPanel.h>
CW_CALLBACK CWReadIntegerSetting(
    CWPluginContext context,
    const char* name,
    long* value);
```

Parameters    Parameters for this function are:

context       Private, opaque IDE state.

name          A C string specifying the name of the XML setting to read.

value          Returns the long integer value of the named setting.

Remarks      `CWReadIntegerSetting` returns the value of a top-level integer XML setting (an integer value contained in the panel's XML stream, but not in a structure or array). If no integer setting by the specified name is found, an error will be returned.

`CWReadIntegerSetting` returns an error if the given setting does not exist or is not of integer type. `CWReadIntegerSetting` should only be called in response to a [reqReadSettings](#) request.

To read a `CWReadIntegerSetting` component of an XML structure or array, use [CWGetIntegerValue](#) instead.

See Also [“reqReadSettings” on page 512](#)

[“CWGetIntegerValue” on page 412](#)

## CWReadRelativePathSetting

Description	Reads a relative path value from the top level of a panel's XML input stream.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWReadRelativePathSetting( <a href="#">CWPluginContext</a> context, const char* name, <a href="#">CWRelativePath</a> * value);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. name            A C string specifying the name of the XML setting to read. value           Returns the relative path value of the named setting.
Remarks	CWReadRelativePathSetting returns the value of a top-level relative path XML setting (a relative path value contained in the panel's XML stream, but not in a structure or array). If no relative path setting by the specified name is found, an error will be returned.  CWReadRelativePathSetting returns an error if the given setting does not exist or is not of relative path type. CWReadRelativePathSetting should only be called in response to a <a href="#">reqReadSettings</a> request.  To read a relative path component of an XML structure or array, use <a href="#">CWGetRelativePathValue</a> instead.
See Also	<a href="#">“reqReadSettings” on page 512</a> <a href="#">“CWGetRelativePathValue” on page 414</a>

## CWReadStringSetting

Description	Reads a string value from the top level of a panel's XML input stream.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWReadStringSetting(

```
CWPluginContext context,  
const char* name,  
const char** value);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	name          A C string specifying the name of the XML setting to read.
	value         Returns a C string containing the value of the named setting. The string returned remains the property of the IDE.

Remarks CWReadStringSetting returns the value of a top-level string XML setting (a string value contained in the panel's XML stream, not in a structure or array). If no string setting by the specified name is found, an error will be returned. The string setting's value is returned in a newly allocated string.

---

**WARNING!** The value string returned remains the property of the IDE, and may be deallocated by the IDE at the end of the pending request. Therefore, to retain the text returned in value globally, be sure to copy it into another string before returning to the IDE.

---

CWReadStringSetting returns an error if the named setting does not exist or is not of string type. CWReadStringSetting should only be called in response to a [reqReadSettings](#) request.

To read a string component of an XML structure or array, use [CWGetStringValue](#) instead.

See Also  
[“reqReadSettings” on page 512](#)  
[“CWGetStringValue” on page 415](#)

## CWSetBooleanValue

Description	Sets the value of a boolean element of an XML array or structure.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWSetBooleanValue( <a href="#">CWPluginContext</a> context,

```
CWSettingID           settingID,  
Boolean                value);
```

Parameters    Parameters for this function are:

context	Private, opaque IDE state.
settingID	Specifies the ID of the setting to change.
value	Specifies the new setting value.

Remarks    CWSetBooleanValue sets the type of an XML setting to boolean, and specifies its value. plug-ins call CWSetBooleanValue in response to a [reqWriteSettings](#) request, to write out the value of a boolean array element or structure field.

If a CWSet...Value call has been previously issued for the setting specified by settingID, its type will be changed to boolean, and its value will be changed to value.

To create a boolean array element setting or structure field setting and obtain its ID, call [CWGetArraySettingElement](#) or [CWGetStructureSettingField](#), respectively. To write out a top-level boolean setting, use [CWWriteBooleanSetting](#).

---

**TIP**    CWSetBooleanValue can be used to change the value of a top-level setting, whose ID was obtained from [CWGetNamedSetting](#). However, it is simpler and easier to call [CWWriteBooleanSetting](#) instead.

---

See Also    [“reqWriteSettings” on page 517](#)

[“CWGetArraySettingElement” on page 408](#)

[“CWGetStructureSettingField” on page 416](#)

[“CWWriteBooleanSetting” on page 477](#)

## **CWSetFloatingPointValue**

Description    Sets the value of a floating point element of an XML array or structure.

Prototype    `#include <CWDropInPanel.h>`

```
CW_CALLBACK CWSetFloatingPointValue(  
    CWPluginContext context,  
    CWSettingID settingID,  
    double value);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	settingID    Specifies the ID of the setting to change.
	value        Specifies the new setting value.
Remarks	<p>CWSetFloatingPointValue sets the type of an XML setting to floating point, and specifies its value. plug-ins call CWSetFloatingPointValue in response to a <a href="#">reqWriteSettings</a> request, to write out the value of a floating point array element or structure field.</p> <p>If a CWSet...Value call has been previously issued for the setting specified by settingID, its type will be changed to floating point, and its value will be changed to value.</p> <p>To create a floating point array element setting or structure field setting and obtain its ID, call <a href="#">CWGetArraySettingElement</a> or <a href="#">CWGetStructureSettingField</a>, respectively. To write out a top-level floating point setting, use <a href="#">CWWriteFloatingPointSetting</a>.</p>
<b>TIP</b>	CWSetFloatingPointValue can be used to change the value of a top-level setting, whose ID was obtained from <a href="#">CWGetNamedSetting</a> . However, it is simpler and easier to call <a href="#">CWWriteFloatingPointSetting</a> instead.
See Also	<a href="#">“reqWriteSettings” on page 517</a> <a href="#">“CWGetArraySettingElement” on page 408</a> <a href="#">“CWGetStructureSettingField” on page 416</a> <a href="#">“CWWriteFloatingPointSetting” on page 477</a>

## CWSetIntegerValue

Description Sets the value of an integer element of an XML array or structure.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWSetIntegerValue(
    CWPluginContext context,
    CWSettingID settingID,
    long value);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

settingID Specifies the ID of the setting to change.

value Specifies the new setting value.

Remarks `CWSetIntegerValue` sets the type of an XML setting to integer, and specifies its value. plug-ins call `CWSetIntegerValue` in response to a [reqWriteSettings](#) request, to write out the value of an integer array element or structure field.

If a `CWSet...Value` call has been previously issued for the setting specified by `settingID`, its type will be changed to integer, and its value will be changed to `value`.

To create an integer array element setting or structure field setting and obtain its ID, call [CWGetArraySettingElement](#) or [CWGetStructureSettingField](#), respectively. To write out a top-level integer setting, use [CWWriteIntegerSetting](#).

---

**TIP** `CWSetIntegerValue` can be used to change the value of a top-level setting, whose ID was obtained from [CWGetNamedSetting](#). However, it is simpler and easier to call [CWWriteIntegerSetting](#) instead.

---

See Also [“reqWriteSettings” on page 517](#)

[“CWGetArraySettingElement” on page 408](#)

[“CWGetStructureSettingField” on page 416](#)

[“CWWriteIntegerSetting” on page 478](#)

## CWSetRelativePathValue

Description Sets the value of a relative path element of an XML array or structure.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWSetRelativePathValue(
    CWPluginContext           context,
    CWSettingID             settingID,
    const CWRelativePath*   value);
```

Parameters Parameters for this function are:

context	Private, opaque IDE state.
settingID	Specifies the ID of the setting to change.
value	Specifies the new setting value.

Remarks CWSetRelativePathValue sets the type of an XML setting to relative path, and specifies its value. plug-ins call CWSetRelativePathValue in response to a [reqWriteSettings](#) request, to write out the value of a relative path array element or structure field.

If a CWSet...Value call has been previously issued for the setting specified by settingID, its type will be changed to relative path, and its value will be changed to value.

To create a relative path array element setting or structure field setting and obtain its ID, call [CWGetArraySettingElement](#) or [CWGetStructureSettingField](#), respectively. To write out a top-level relative path setting, use [CWWriteRelativePathSetting](#).

---

**TIP** CWSetRelativePathValue can be used to change the value of a top-level setting, whose ID was obtained from [CWGetNamedSetting](#). However, it is simpler and easier to call [CWWriteRelativePathSetting](#) instead.

---

See Also [“reqWriteSettings” on page 517](#)

[“CWGetArraySettingElement” on page 408](#)

[“CWGetStructureSettingField” on page 416](#)

[“CWWriteRelativePathSetting” on page 479](#)

## CWSetStringValue

Description Sets the value of a string element of an XML array or structure.

Prototype

```
#include <CWDropInPanel.h>
CW_CALLBACK CWSetStringValue(
    CWPluginContext context,
    CWSettingID settingID,
    const char* value);
```

Parameters Parameters for this function are:

context	Private, opaque IDE state.
settingID	Specifies the ID of the setting to change.
value	Specifies the new setting value.

Remarks CWSetStringValue sets the type of an XML setting to string, and specifies its value. plug-ins call CWSetStringValue in response to a [reqWriteSettings](#) request, to write out the value of a string array element or structure field.

If a CWSet...Value call has been previously issued for the setting specified by settingID, its type will be changed to string, and its value will be changed to value.

To create a string array element setting or structure field setting and obtain its ID, call [CWGetArraySettingElement](#) or [CWGetStructureSettingField](#), respectively. To write out a top-level string setting, use [CWWriteStringSetting](#).

---

**TIP** CWSetStringValue can be used to change the value of a top-level setting, whose ID was obtained from [CWGetNamedSetting](#). However, it is simpler and easier to call [CWWriteStringSetting](#) instead.

---

See Also [“reqWriteSettings” on page 517](#)

[“CWGetArraySettingElement” on page 408](#)

[“CWGetStructureSettingField” on page 416](#)

[“CWWriteStringSetting” on page 479](#)

## CWWriteBooleanSetting

Description	Writes a boolean value to the top level of a panel's exported XML data.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWWriteBooleanSetting( <a href="#"><u>CWPluginContext</u></a> context, const char*         name, Boolean             value);
Parameters	Parameters for this function are:  context         Private, opaque IDE state. name             Specifies the name of the top-level setting to create, as a C string. value             Specifies the new setting value.
Remarks	CWWriteBooleanSetting writes the value of a top-level boolean setting to the plug-in's XML output stream. CWWriteBooleanSetting should only be called in response to a <a href="#"><u>reqWriteSettings</u></a> request.  To write a boolean component of an XML structure or array, use <a href="#"><u>CWPanlSetBooleanValue</u></a> instead.
See Also	<a href="#">“reqWriteSettings” on page 517</a> <a href="#">“CWSetBooleanValue” on page 471</a>

## CWWriteFloatingPointSetting

Description	Writes a floating point value to the top level of a panel's exported XML data.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWWriteFloatingPointSetting( <a href="#"><u>CWPluginContext</u></a> context, const char*         name,

```
double value);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	name          Specifies the name of the top-level setting to create, as a C string.
	value         Specifies the new setting value.
Remarks	CWWriteFloatingPointSetting writes the value of a top-level floating point setting to the plug-in's XML output stream. CWWriteFloatingPointSetting should only be called in response to a <a href="#">reqWriteSettings</a> request.  To write a floating point component of an XML structure or array, use <a href="#">CWPanlSetFloatingPointValue</a> instead.
See Also	<a href="#">“reqWriteSettings” on page 517</a> <a href="#">“CWSetFloatingPointValue” on page 472</a>

## CWWriteIntegerSetting

Description	Writes an integer value to the top level of a panel's exported XML data.
Prototype	#include <CWDropInPanel.h> CW_CALLBACK CWWriteIntegerSetting( <a href="#">CWPluginContext</a> context, const char* name, long value);
Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	name          Specifies the name of the top-level setting to create, as a C string.
	value         Specifies the new setting value.
Remarks	CWWriteIntegerSetting writes the value of a top-level integer setting to the plug-in's XML output stream. CWWriteIntegerSetting should only be called in response to a <a href="#">reqWriteSettings</a> request.

To write a integer component of an XML structure or array, use [CWPanlSetIntegerValue](#) instead.

See Also [“reqWriteSettings” on page 517](#)

[“CWSetIntegerValue” on page 474](#)

## **CWWriteRelativePathSetting**

Description Writes a relative path value to the top level of a panel's exported XML data.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWWriteRelativePathSetting(
    CWPluginContext           context,
    const char*                  name,
    const CWRelativePath*   value);
```

Parameters Parameters for this function are:

context Private, opaque IDE state.

name Specifies the name of the top-level setting to create, as a C string.

value Specifies the new setting value.

Remarks *CWWriteRelativePathSetting* writes the value of a top-level relative path setting to the plug-in's XML output stream. *CWwriteRelativePathSetting* should only be called in response to a [reqWriteSettings](#) request.

To write a relative path component of an XML structure or array, use [CWPanlSetRelativePathValue](#) instead.

See Also [“reqWriteSettings” on page 517](#)

[“CWSetRelativePathValue” on page 475](#)

## **CWWriteStringSetting**

Description Writes a string value to the top level of a panel's exported XML data.

Prototype 

```
#include <CWDropInPanel.h>
CW_CALLBACK CWWriteStringSetting(
```

## Settings Panel Plug-in API Reference

### User Routines for Settings Panel Plug-ins

---

```
CWPluginContext context,
const char* name,
const char* value);
```

Parameters	Parameters for this function are:
	context      Private, opaque IDE state.
	name          Specifies the name of the top-level setting to create, as a C string.
	value         Specifies the new setting value.
Remarks	<p>CWWriteStringSetting writes the value of a top-level string setting to the plug-in's XML output stream.</p> <p>CWWriteStringSetting should only be called in response to a <a href="#">reqWriteSettings</a> request.</p> <p>To write a string component of an XML structure or array, use <a href="#">CWPanlSetStringValue</a> instead.</p>
See Also	<a href="#">“reqWriteSettings” on page 517</a> <a href="#">“CWSetStringValue” on page 476</a>

## User Routines for Settings Panel Plug-ins

This section documents informational entry points specific to panel plug-ins.

The routines documented in this section are:

- [CWPlugin\\_GetHelpInfo Entry Point](#)

### **CWPlugin\_GetHelpInfo Entry Point**

Description    An optional entry point implemented by Windows plug-ins to inform the IDE about help information associated with a panel.

Prototype    

```
#include <CWDropInPanel.h>
CWPLUGIN_ENTRY (CWPlugin_GetHelpInfo)
    (const CWHelpInfo** helpInfo);
```

Parameters    Parameters for this function are:

	helpInfo	The plug-in returns a pointer to help information.
Remarks		Windows plug-ins use this entry point to specify the name of a WinHelp help file to be displayed when the user clicks <b>Help</b> in the <b>Target Settings</b> dialog. This entry point is optional.  The IDE searches for plug-in help files in the <code>Help</code> directory within the CodeWarrior IDE install directory.
See Also		<a href="#">“Help Information Entry Point” on page 375</a> <a href="#">“CWHelpInfo” on page 482</a>

## Data Structures for Settings Panel Plug-ins

This section discusses the following topics:

- [CWDIALOG](#)
- [CWHelpInfo](#)
- [CWSettingID](#)
- [PanelFlags](#)
- [PanelParamBlkPtr](#)
- [PanelParameterBlock](#)

### CWDIALOG

Description	Specifies the type of a Mac OS panel dialog.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; #ifndef CW_STRICT_DIALOGS     typedef struct DummyDialog* CWDIALOG; #else     typedef DialogPtr                  CWDIALOG; #endif</pre>
Remarks	Depending on the definition of <code>CW_STRICT_DIALOGS</code> , this data type is equivalent to an opaque type or to the Mac OS <code>DialogPtr</code> type.

All new Mac OS panel plug-ins should be compiled with [CW\\_STRICT\\_DIALOGS](#) enabled, and should not use the `dialog` member of a [PanelParameterBlock](#).

See Also [“CW\\_STRICT\\_DIALOGS” on page 494](#)

[“PanelParameterBlock” on page 485](#)

## **CWHelpInfo**

Description Describes a Windows panel’s associated help file to the IDE.

Prototype

```
#include <CWPlugins.h> /* Windows */
typedef struct CWHelpInfo
{
    short           version;
    const char *   helpFileName;
} CWHelpInfo;
```

Fields The fields in this structure are:

---

<b>version</b>	Specifies the version of the CWHelpInfo structure. plug-ins should set this to <a href="#">kCurrentCWHelpInfoVersion</a> .
<b>helpFileName</b>	Specifies the name of a settings panel’s associated WinHelp file as a C string.

---

Remarks Windows panels optionally return a pointer to a CWHelpInfo structure to tell the IDE the name of any associated help file. The help file name should end with a '.hlp' extension. This help file is displayed by the IDE when the user clicks the **Help** button in the **Target Settings** dialog, and should be a standard WinHelp file.

See Also [“Help Information Entry Point” on page 375](#)

[“kCurrentCWHelpInfoVersion” on page 495](#)

## **CWSettingID**

Description Identifies a container for a simple XML data type, an XML array type, or an XML structure type.

Prototype

```
#include <CWDropInPanel.h> /* Windows */
```

```
#include <CWDropInPanel.h>      /* Mac OS */
typedef struct MWSetting* CWSettingID;
```

**Remarks** A CWSettingID represents an XML setting container, associating an XML name with a data value of a particular type. CWSettingIDs can store array, structure, and simple types.

Plug-ins call [CWGetNamedSetting](#) to obtain CWSettingIDs for simple types. Plug-ins call [CWGetStructureSettingField](#) to obtain CWSettingIDs for structure fields, and [CWGetArraySettingElement](#) to obtain CWSettingIDs for array elements.

**See Also** [“CWGetNamedSetting” on page 413](#)

[“CWGetStructureSettingField” on page 416](#)

[“CWGetArraySettingElement” on page 408](#)

## PanelFlags

**Description** Describes a panel plug-in's capabilities to the IDE.

**Prototype**

```
#include <CWDropInPanel.h> /* Windows */
#include <CWDropInPanel.h>      /* Mac OS */
```

```
typedef struct PanelFlags {
    unsigned short    rsrcversion;
    CWDataType        dropintype;
    unsigned short    earliestCompatibleAPIVersion;
    unsigned long     dropinflags;
    CWDataType        panelfamily;
    unsigned short    newestAPIVersion;
    unsigned short    dataversion;
    unsigned short    panelscope;
} PanelFlags;
```

**Fields** The fields in this structure are:

---

<b>rsrcversion</b>	Specifies the version of the PanelFlags structure.
--------------------	--

<b>dropintype</b>	Specifies the plug-in type (should be 'PanL' for settings panels).
-------------------	--

**earliestCompatibleAPIVersion**

Specifies the earliest API version a plug-in is compatible with.

**dropinflags**

Specifies the capabilities of a plug-in. Use a combination of [usesStrictAPI](#), [supportsByteSwapping](#), and [supportsTextSettings](#).

**panelfamily**

Specifies the family of a plug-in. See [Standard settings panel family type codes](#) for appropriate values.

**newestAPIVersion**

Specifies the most recent API version a plug-in is compatible with. Most plug-ins should use [DROPINPANELAPIVERSION](#).

**dataversion**

Specifies the version number of a panel's settings data.

**panelscope**

Specifies the scope of a plug-in, which determines which CodeWarrior IDE dialog the plug-in appears in. Use one of [panelScopeGlobal](#), [panelScopeProject](#), or [panelScopeTarget](#).

---

**Remarks**

The `PanelFlags` structure reports information to the IDE about a plug-in's capabilities. The `CWPlugin_GetDropInFlags` entry point for a panel plug-in returns a `PanelFlags` structure. See [Specifying Preference Panel Capabilities](#) for more information about returning a `PanelFlags` structure to the IDE.

**See Also**

[“Specifying Preference Panel Capabilities” on page 68](#)

[“Specifying Panel Family” on page 73](#)

[“PanelFlags Structure” on page 340](#)

[“Settings Panel Flags” on page 341](#)

## PanelParamBlkPtr

Description	A pointer to a <a href="#">PanelParameterBlock</a> .
Prototype	#include <CWDropInPanel.h> typedef PanelParameterBlock* PanelParamBlkPtr;
Remarks	PanelParamBlkPtr is a pointer to a panel parameter block, <a href="#">PanelParameterBlock</a> . The IDE passes a PanelParamBlkPtr to the panel plug-in to communicate information to the panel.
See Also	<a href="#">“PanelParameterBlock” on page 485</a>

## PanelParameterBlock

Description	Contains information passed to the plug-in by the IDE, and required by the IDE when servicing calls made by panel plug-ins. {temp
Prototype	<pre>#include &lt;CWDropInPanel.h&gt; /* parameter block -- this is passed to the dropin at each request */  typedef struct PanelParameterBlock {     /* common to all dropins */     long    request;         /* [-&gt;] requested action (see below)      */     long    version;         /* [-&gt;] version # of shell's API          */     void    *context;         /* [-&gt;] reserved for use by shell          */     void    *storage;         /* [&lt;-&gt;] reserved for use by the dropin   */     FSSpec  targetfile;         /* [-&gt;] FSSpec of current project         */      /* specific to panels */     CWDDialog    dialog;         /* [-&gt;] pointer to PreferencesÉ dialog     */     Handle       originalPrefs;         /* [-&gt;] panel's original options data      */     Handle       currentPrefs;         /* [&lt;-&gt;] panel's current options data      */     Handle       factoryPrefs;         /* [&lt;-&gt;] panel's "factory" options data    */ }</pre>

```
EventRecord *event;
/* [->] dialog event (for reqFilterEvent) */
short      baseItems;
/* [->] # of items in dialog shell           */
short      itemHit;
/* [<->] for reqFilterEvent and reqItemHit */
Boolean    canRevert;
/* [<-] enable Revert button                  */
Boolean    canFactory;
/* [<-] enable Factory button                */
Boolean    reset;
/* [<-] access paths must be reset          */
Boolean    recompile;
/* [<-] files must be recompiled            */
Boolean    relink;
/* [<-] project must be relinked           */
AEKeyword  prefsKeyword;
/* [->] for reqAEGetPref and reqAESetPref */
AEDesc     prefsDesc;
/* [->] for reqAESetPref                   */
Boolean    debugOn;
/* [->] turning on debugging?              */
FSSpec    oldtargfile;
/* [->] previous project file FSSpec       */

/* version 2 API */
CWPanelCallbacks*      callbacks;

/* version 3 API */
Boolean    reparse;           /* [<-] project must
be reparsed   */

/* version 4 API */
DragReference dragref;
/* [->] for drag-related requests   */
Rect      dragrect;
/* [<-] rect to track mouse in      */
Point    dragmouse;
/* [->] mouse location during drag */

/* version 5 API */
unsigned char  toEndian;
```

```
/* [->] for reqByteSwapData, the
   endian we are swapping to */

/* CWPro 3 temporary placeholders for opaque
   references to prefs data. These will be removed in
   Pro 4. */
CWMemHandle      originalPrefsMemHandle;
CWMemHandle      currentPrefsMemHandle;
CWMemHandle      factoryPrefsMemHandle;
CWMemHandle      panelPrefsMemHandle;

/* version 11 api */
long listViewCellRow;
   /* [->] the cell row of the listView */
long listViewCellCol;
   /* [->] the cell col of the listView */
/*
   These fields are supported only from
DropinPanelPrefVersion 12 and higher for long
filename support. Older plug-ins will get will use
FSSpec based fields targetFile and oldTargetFile

Since the IDE is LFN enabled, the will attempt
to convert the internal LFN data struture to
create a valid FSSpec. If the API fails to create a
valid FSSpec, the FSSpec name field would be
filled with the project name.

The newer plug-ins or those plug-ins that
decide to upgrade to the newer library(PluginLib5)
will have to be aware that these fields are the new
means of accessing the target file and old target
file. The older fsspec based fields will be
deprecated for the newer plug-ins (version 12 and
higher) and will be zeroed out.
*/
/* version 12 api */
CWPanelFileSpec lfnTargetFile;
CWPanelFileSpec lfnOldTargetFile;

} PanelParameterBlock, *PanelParameterBlockPtr;
```

Fields    The fields in *PanelParameterBlock* are:

request	Specifies the action requested of a panel by the IDE. Also determines which of the remaining parameter block fields are required to service the request.
version	Specifies the current version of the panel API.
context	Reserved for use by the IDE. Do not modify this field.
storage	Reserved for use by the panel drop-in. The IDE guarantees that the contents of this field will be preserved between <a href="#">reqInitPanel</a> and <a href="#">reqTermPanel</a> requests to the panel. Most plug-ins should <i>not</i> use this field. See important warning below.
targetfile	Specifies the location of the current project file.
dialog	Contains a pointer to the IDE's <b>Target Settings</b> dialog box for plug-ins that clear the <a href="#">usesStrictAPI</a> flag. All new plug-ins should set this flag, and be compiled with <a href="#">CW_STRICT_DIALOGS</a> set to true, and so this field should not be used.
originalPrefs	Contains a copy of the most recently saved settings data. Plug-ins should not modify this field.
currentPrefs	Contains the current panel settings data. Plug-ins are expected to modify this field.
factoryPrefs	Contains the default settings data for the panel. Plug-ins modify this field when the IDE sends the <a href="#">reqGetFactory</a> request.

event	Contains a copy of the most recently received (not yet handled) Mac OS event. Plug-ins which implement user items (special controls) may need to examine and possibly modify this field when responding to a <a href="#">reqFilter</a> request.
baseItems	Indicates the number of items in the host dialog, prior to addition of the panel's controls. Plug-ins should subtract this number from itemHit when responding to <a href="#">reqItemHit</a> , <a href="#">reqDrawCustomItem</a> , <a href="#">reqActivateItem</a> , <a href="#">reqDeactivateItem</a> , <a href="#">reqHandleKey</a> , and <a href="#">reqHandleClick</a> requests. The resulting adjusted item number corresponds to item numbers in the panel's original 'PPop' item list.
itemHit	Specifies the number of the item with which the user has just finished interacting. Also used to specify <b>Edit</b> menu items, for <a href="#">reqFindStatus</a> and <a href="#">reqObeyCommand</a> requests. Plug-ins can modify this value to simulate hits in other items
canRevert	Plug-ins set this flag when responding to a <a href="#">reqItemHit</a> request, to indicate whether the current settings differ from the most recently saved (original) settings. When set true, the IDE enables the <b>Revert</b> button in the <b>Target Settings</b> dialog.
canFactory	Plug-ins set this flag when responding to a <a href="#">reqItemHit</a> request, to indicate whether the current settings differ from the factory default settings. When set true, the IDE enables the <b>Factory Settings</b> button in the <b>Target Settings</b> dialog.
reset	Plug-ins set this flag when responding to a <a href="#">reqValidate</a> request, to indicate whether the current settings require the IDE to re-scan for project files in the project's access paths. When set true, the IDE re-scans for files.

---

recompile	Plug-ins set this flag when responding to a <a href="#">reqValidate</a> request, to indicate whether the current settings require the IDE to recompile all files in the project. When set true, the IDE recompiles the project.
relink	Plug-ins set this flag when responding to a <a href="#">reqValidate</a> request, to indicate whether the current settings require the IDE to re-link the target executable. When set true, the IDE re-links the project.
prefsKeyword	Specifies the Apple Event keyword for the pending Apple Event as a four-character code.
prefsDesc	Contains the data for the pending Apple Event, in an Apple Event descriptor. The plug-in is responsible for disposing this when responding to a <a href="#">reqAESetPref</a> request, and for allocating it when responding to a <a href="#">reqAEGetPref</a> request.
debugOn	Specifies whether the user has enabled debugging in the <b>Project</b> menu.
oldtargfile	Specifies the previous name and location of the current project file, prior to being renamed.
reparse	Set by a plug-in if the project's files must be reprocessed by the class browser. This field is currently unused.
dragref	Specifies the Drag Manager drag reference for the current drag operation. The panel uses this value when making calls to the Drag Manager.
dragrect	Specifies the rectangle that the IDE treats as the drag destination.
dragmouse	Specifies the current mouse location during a drag operation.

`toEndian`      Specifies the byte ordering to which setting data should be converted during a [reqByteSwapData](#) request (or the *current* byte ordering of the settings data, during a [reqFirstLoad](#) request). Contains either 1 for “big endian” or 2 for “little endian.”

`originalPrefsMemHandle`

Obsolete. Do not use.

`currentPrefsMemHandle`

Obsolete. Do not use.

`factoryPrefsMemHandle`

Obsolete. Do not use.

`panelPrefsMemHandle`

Obsolete. Do not use.

`listViewCellRow`

Specifies the row within a list view panel.

`listViewCellCol`

Specifies the column within a list view panel.  
(The comment in the header file is incorrect - it's the column, as the name implies, not the row.)

`lfnTargetFile`

Specifies the new file name for long filename support. “lfn” stands for “long file name.”

`lfnOldTargetFile`

Specifies the old file name for long filename support. “lfn” stands for “long file name.”

**Remarks**      The CodeWarrior IDE issues commands to Mac OS settings panel plug-ins by calling the panel and passing a parameter block data structure, `PanelParameterBlock` to the plug-in’s `main()` entry

point. The panel parameter block contains information about the kind of action the panel should perform and other information that the panel might need when servicing the IDE requests.

Not all fields of the parameter block are guaranteed to be valid for every request. See individual requests for details of field validity.

**WARNING!**

The storage field is shared between all instances of the same plug-in. That is, the IDE provides one copy of the storage field per plug-in, not one per instance of a panel. Thus, if two projects using a settings panel plug-in are open simultaneously, the storage field will be shared by both panels. This can lead to problems accessing global storage. In practice, plug-ins are advised not to use the storage field.

---

See Also

[“PanelParameterBlock” on page 485](#)

## Constants for Settings Panel Plug-ins

The predefined symbols for settings panels are:

- [CW\\_STRICT\\_DIALOGS](#)
- [DROPINPANELAPIVERSION](#)
- [menu\\_Clear](#)
- [menu\\_Copy](#)
- [menu\\_Cut](#)
- [menu\\_Paste](#)
- [menu\\_SelectAll](#)
- [panelScopeGlobal](#)
- [panelScopeProject](#)
- [panelScopeTarget](#)
- [reqActivateItem](#)
- [reqAEGetPref](#)
- [reqAESetPref](#)
- [reqByteSwapData](#)
- [reqDeactivateItem](#)

- [reqDragDrop](#)
- [reqDragEnter](#)
- [reqDragExit](#)
- [reqDragWithin](#)
- [reqDrawCustomItem](#)
- [reqFilter](#)
- [reqFindStatus](#)
- [reqFirstLoad](#)
- [reqGetData](#)
- [reqGetFactory](#)
- [reqHandleClick](#)
- [reqHandleKey](#)
- [reqInitDialog](#)
- [reqInitPanel](#)
- [reqItemHit](#)
- [reqObeyCommand](#)
- [reqPutData](#)
- [reqReadSettings](#)
- [reqRenameProject](#)
- [reqSetupDebug](#)
- [reqTermDialog](#)
- [reqTermPanel](#)
- [reqUpdatePref](#)
- [reqValidate](#)
- [reqWriteSettings](#)
- [supportsByteSwapping](#)
- [supportsTextSettings](#)
- [usesStrictAPI](#)

## **CW\_STRICT\_DIALOGS**

Description	A compile-time switch that controls whether a Mac OS hosted settings panel can use the Mac OS dialog manager.
Prototype	#include <CWDropInPanel.h> #define CW_STRICT_DIALOGS 0
Remarks	When CW_STRICT_DIALOGS is defined as any nonzero value, settings panels for a Mac OS host may not utilize Mac OS dialog manager routines to manipulate settings panel controls. Instead, a panel plug-in must rely entirely on the settings panel API to draw, get, and set dialog items.
	CW_STRICT_DIALOGS controls the type of the dialog field in the <a href="#">PanelParameterBlock</a> passed to the plug-in. When true, dialog is an anonymous pointer type. When set to zero, dialog is a Mac OS DialogPtr. This define controls compile-time syntactic checks, not runtime behavior of the IDE.
	CW_STRICT_DIALOGS should be set true (nonzero) for all current and future Mac OS plug-ins. New versions of the IDE no longer support settings panels compiled with CW_STRICT_DIALOGS defined as 0.
<b>NOTE</b>	All Mac OS settings panels should define CW_STRICT_DIALOGS as true, and should also set the <a href="#">usesStrictAPI</a> bit in their dropinflags.
See Also	<a href="#">“Settings Panel Flags” on page 341</a>

## **DROPINPANELAPIVERSION**

Description	Specifies the current version of the settings panel plug-in API at compile time.
Prototype	#include <CWDropInPanel.h> /* Windows */ #include <CWDropInPanel.h> /* Mac OS */
Remarks	This predefined symbol specifies the current version of the settings panel plug-in API. This value is most useful in specifying compatible IDE versions in the <a href="#">PanelFlags</a> structure returned by a panel’s CWPlugin_GetDropInFlags entry point.

See Also    [“Specifying Preference Panel Capabilities” on page 68](#)

## **kCurrentCWHelpInfoVersion**

Description	Indicates the current version of a <a href="#">CWHelpInfo</a> structure.
Prototype	#include "CWPlugins.h"
Remarks	Windows plug-ins should set the <code>version</code> field of a <a href="#">CWHelpInfo</a> structure to this value.
See Also	<a href="#">“CWHelpInfo” on page 482</a> <a href="#">“CWPlugin_GetHelpInfo Entry Point” on page 480</a>

## **menu\_Clear**

Description	Represents the <b>Clear</b> command in the <b>Edit</b> menu.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this value to Mac OS panel plug-ins during <a href="#">reqFindStatus</a> and <a href="#">reqObeyCommand</a> requests in the <code>itemHit</code> field of the <a href="#">PanelParameterBlock</a> , to indicate the relevant <b>Edit</b> menu item.
See Also	<a href="#">“Handling Edit Menu Commands” on page 393</a> <a href="#">“reqFindStatus” on page 505</a> <a href="#">“reqObeyCommand” on page 511</a>

## **menu\_Copy**

Description	Represents the <b>Copy</b> command in the <b>Edit</b> menu.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this value to Mac OS panel plug-ins during <a href="#">reqFindStatus</a> and <a href="#">reqObeyCommand</a> requests in the <code>itemHit</code> field of the <a href="#">PanelParameterBlock</a> , to indicate the relevant <b>Edit</b> menu item.
See Also	<a href="#">“Handling Edit Menu Commands” on page 393</a> <a href="#">“reqFindStatus” on page 505</a>

[“reqObeyCommand” on page 511](#)

## menu\_Cut

Description	Represents the <b>Cut</b> command in the <b>Edit</b> menu.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this value to Mac OS panel plug-ins during <a href="#">reqFindStatus</a> and <a href="#">reqObeyCommand</a> requests in the <code>itemHit</code> field of the <a href="#">PanelParameterBlock</a> , to indicate the relevant <b>Edit</b> menu item.
See Also	<a href="#">“Handling Edit Menu Commands” on page 393</a> <a href="#">“reqFindStatus” on page 505</a> <a href="#">“reqObeyCommand” on page 511</a>

## menu\_Paste

Description	Represents the <b>Paste</b> command in the <b>Edit</b> menu.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this value to Mac OS panel plug-ins during <a href="#">reqFindStatus</a> and <a href="#">reqObeyCommand</a> requests in the <code>itemHit</code> field of the <a href="#">PanelParameterBlock</a> , to indicate the relevant <b>Edit</b> menu item.
See Also	<a href="#">“Handling Edit Menu Commands” on page 393</a> <a href="#">“reqFindStatus” on page 505</a> <a href="#">“reqObeyCommand” on page 511</a>

## menu\_SelectAll

Description	Represents the <b>Select All</b> command in the <b>Edit</b> menu.
Prototype	#include <CWDropInPanel.h> enum {
Remarks	The IDE sends this value to Mac OS panel plug-ins during <a href="#">reqFindStatus</a> and <a href="#">reqObeyCommand</a> requests in the <code>itemHit</code>

field of the [PanelParameterBlock](#), to indicate the relevant **Edit** menu item.

See Also     [“Handling Edit Menu Commands” on page 393](#)  
              [“reqFindStatus” on page 505](#)  
              [“reqObeyCommand” on page 511](#)

## panelScopeGlobal

Description	Indicates that a settings panel should appear in the <b>Edit &gt; Preferences</b> dialog.
Prototype	#include <CWDropInPanel.h>
Remarks	Settings panels return this value in the <code>panelscope</code> field of the <a href="#">PanelFlags</a> structure, from their <code>CWPlugin_GetDropInFlags</code> entry point.  Settings panels which specify a scope of <code>panelScopeGlobal</code> appear in the <b>Preferences</b> dialog, and are always enabled, regardless of whether a project is open.
See Also	<a href="#">“PanelFlags Structure” on page 340</a>

## panelScopeProject

Description	Indicates that a settings panel should appear in the <b>Edit &gt; Version Control Settings</b> dialog.
Prototype	#include <CWDropInPanel.h>
Remarks	Settings panels return this value in the <code>panelscope</code> field of the <a href="#">PanelFlags</a> structure, from their <code>CWPlugin_GetDropInFlags</code> entry point.  Settings panels which specify a scope of <code>panelScopeProject</code> appear in the <b>Version Control Settings</b> dialog, and are always enabled, regardless of whether a project is open. Currently, this scope should only be used by version control plug-ins.
See Also	<a href="#">“PanelFlags Structure” on page 340</a>

## panelScopeTarget

Description	Indicates that a settings panel should appear in the <b>Edit &gt; Target Settings</b> dialog.
Prototype	#include <CWDropInPanel.h>
Remarks	Settings panels return this value in the <code>panelscope</code> field of the <a href="#">PanelFlags</a> structure, from their <code>CWPlugin_GetDropInFlags</code> entry point.
	Settings panels which specify a scope of <code>panelScopeTarget</code> appear in the <b>Target Settings</b> dialog, and are enabled if at least one project is open (and that project uses plug-ins associated with the settings panel). This is the scope used by panels associated with compilers and linkers.
See Also	<a href="#">“PanelFlags Structure” on page 340</a>

## reqActivateItem

Description	Requests that a Mac OS panel redraw a custom dialog item to show it has input focus.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request when a custom dialog item acquires input focus. Plug-ins should respond by highlighting the custom item.
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the current settings data</li><li>• <code>itemHit</code> contains the number of the custom dialog item to highlight</li><li>• <code>baseItems</code> contains the number of control items that the IDE has already placed in the <b>Target Settings</b> dialog box</li></ul>
See Also	<a href="#">“Managing Input Focus” on page 373</a> <a href="#">“Managing Input Focus (Mac OS)” on page 393</a>

## reqAEGetPref

Description	Requests that a Mac OS panel return the value of a setting in response to an Apple Event.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE asks the panel to access an item of settings information based on an Apple Event.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• currentPrefs contains the settings data to extract the item from</li><li>• prefsKeyword contains the 4 character (32-bit) Apple Event keyword specifying the setting to return</li></ul> <p>On exit:</p> <ul style="list-style-type: none"><li>• prefsDesc contains an Apple Event descriptor containing the data of the requested setting (the plug-in is responsible for allocating this using the Mac OS toolbox routine AECreateDesc)</li></ul>
See Also	<a href="#">“Handling Apple Events” on page 396</a> <a href="#">“Apple Event Dictionary ('aete') Resource” on page 387</a>

## reqAESetPref

Description	Requests that a Mac OS panel change the value of a setting in response to an Apple Event.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE asks the panel to store an item of settings information based on an Apple Event.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• currentPrefs contains the settings data to operate on</li><li>• prefsKeyword contains the 4 character (32-bit) Apple Event keyword specifying the setting to modify</li><li>• prefsDesc contains an Apple Event descriptor containing the new data for the specified setting</li></ul> <p>On exit:</p>

- `currentPrefs` contains the changed settings data.

See Also [“Handling Apple Events” on page 396](#)

[“Apple Event Dictionary \('aete'\) Resource” on page 387](#)

## reqByteSwapData

Description Asks the panel to adjust its settings data for endian-ness.

Prototype `#include <CWDropInPanel.h>`

Remarks The IDE sends this request to adjust a panel's settings data to little-endian or big-endian format.

Mac OS On entry:

- `currentPrefs` contains the settings data to operate upon
- `toEndian` indicates the endian-ness the plug-in must convert its data to

On exit:

- `currentPrefs` contains the swapped settings data

See Also [“reqByteSwapData Request” on page 359](#)

## reqDeactivateItem

Description Requests that a Mac OS panel redraw a custom dialog item that no longer has input focus.

Prototype `#include <CWDropInPanel.h>`

Remarks The IDE sends this request when a custom dialog item loses input focus. plug-ins should respond by un-highlighting the custom dialog item.

Mac OS On entry

- `currentPrefs` contains a handle to a copy of the panel's current settings data
- `itemHit` contains the dialog item to draw without the input focus
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

See Also [“Managing Input Focus” on page 373](#)

[“Managing Input Focus \(Mac OS\)” on page 393](#)

## reqDragDrop

Description Informs a Mac OS panel that a dragged item has been dropped on it.

Prototype `#include <CWDropInPanel.h>`

Remarks The IDE informs the panel that the user has dropped an object on an item. A panel normally responds by extracting data from the drop using Drag Manager routines, and by updating its current settings data.

Mac OS On entry:

- `dragref` contains the Drag Manager drag reference value of the current drag operation
- `dragmouse` contains the mouse location, in local coordinates
- `dialog` contains a pointer to the **Target Settings** dialog
- `currentPrefs` contains the settings data to operate on
- `itemHit` contains the dialog item that the user has dragged the object onto
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

On exit:

- if the panel sets `itemHit` to a non-zero value, the IDE will send a [reqItemHit](#) request to the panel with `itemHit` containing the item to act on.

See Also [“Handling Drag And Drop” on page 395](#)

[“reqDragEnter” on page 502](#)

[“reqDragExit” on page 503](#)

[“reqDragWithin” on page 504](#)

## **reqDragEnter**

**Description** Informs a Mac OS panel that a dragged item has been dragged, but not dropped, onto it.

**Prototype** #include <CWDropInPanel.h>

**Remarks** The IDE informs the panel that the user has just dragged, but not dropped, an object onto one of its items. The panel should call the appropriate Drag Manager routines to handle user feedback if the item can receive the object being dragged.

**Mac OS** On entry:

- `dragref` contains the Drag Manager drag reference value of the current drag operation
- `dragmouse` contains the mouse location, in local coordinates
- `dialog` contains a pointer to the **Target Settings** dialog
- `currentPrefs` contains the settings data to operate on
- `itemHit` contains the dialog item that the user has dragged the object onto
- `baseItems` contains the number of control items that the IDE has already placed in the Target Settings dialog box

On exit:

- `dragrect` optionally contains new dimensions that delimit the area that the item responds to subsequent drag requests

Typically a panel plug-in will decrease the values in `dragrect` to ignore scroll bars, or increase `dragrect` to allow for autoscrolling.

The plug-in also returns an error code, which the IDE interprets as follows:

- `noErr` means the object can be dropped onto the item and the IDE can continue to send [reqDragWithin](#), [reqDragExit](#), and [reqDragDrop](#) requests for this drag operation
- `dragNotAcceptedErr` means the item supports dragging and dropping, but not objects of the kind that is currently being dragged and that the IDE should not send any more drag requests for the current drag operation

- paramErr means the item does not support dragging and dropping and the IDE should not send any more drag requests for the current drag operation

See Also

[“Handling Drag And Drop” on page 395](#)

[“reqDragDrop” on page 501](#)

[“reqDragEnter” on page 502](#)

[“reqDragWithin” on page 504](#)

## reqDragExit

Description      Informs a Mac OS panel that a dragged item has been dragged, but not dropped, out of it.

Prototype      `#include <CWDropInPanel.h>`

Remarks      The IDE informs the panel that the user has dragged an object out of an item. The panel should call appropriate Drag Manager routines to handle user feedback for the item.

Mac OS      On entry:

- dragref contains the Drag Manager drag reference value of the current drag operation
- dragmouse contains the mouse location, in local coordinates
- dialog contains a pointer to the **Target Settings** dialog
- currentPrefs contains the settings data to operate on
- itemHit contains the dialog item that the user has dragged the object onto
- baseItems contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

See Also

[“Handling Drag And Drop” on page 395](#)

[“reqDragDrop” on page 501](#)

[“reqDragEnter” on page 502](#)

[“reqDragWithin” on page 504](#)

## **reqDragWithin**

Description	Informs a Mac OS panel that a dragged item has been dragged, but not dropped, within it.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE informs the panel that the user has dragged, but not dropped, an object within an item. The IDE sends this request to allow the panel to track the user's mouse movements during a drag operation, and to perform any additional drop point highlighting that may be required.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>dragref</code> contains the Drag Manager drag reference value of the current drag operation</li><li>• <code>dragmouse</code> contains the mouse location, in local coordinates</li><li>• <code>dialog</code> contains a pointer to the <b>Target Settings</b> dialog</li><li>• <code>currentPrefs</code> contains the settings data to operate on</li><li>• <code>itemHit</code> contains the dialog item that the user has dragged the object onto</li><li>• <code>baseItems</code> contains the number of control items that the IDE has already placed in the <b>Target Settings</b> dialog box</li></ul>

See Also [“Handling Drag And Drop” on page 395](#)

[“reqDragDrop” on page 501](#)

[“reqDragEnter” on page 502](#)

[“reqDragExit” on page 503](#)

## **reqDrawCustomItem**

Description	Asks a Mac OS panel to draw a custom dialog item.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE asks the panel to draw a custom dialog box item. The panel should save and restore any port characteristics it modifies.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>itemHit</code> contains the dialog item to draw</li></ul>

- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

See Also [“Drawing Custom Items” on page 392](#)

## reqFilter

Description Asks a Mac OS panel to respond to a low-level Mac OS event.

Prototype `#include <CWDropInPanel.h>`

Remarks The IDE asks the panel to process a Mac OS event. The panel can use this call to handle custom dialog controls.

Mac OS On entry:

- `event` contains a Mac OS event
- `dialog` contains a pointer to the **Target Settings** dialog box
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

On exit:

- `itemHit` contains the number of the item in the **Target Settings** dialog box that received the event “hit”

See Also [“Filtering Low-Level Events” on page 391](#)

## reqFindStatus

Description Asks a Mac OS panel to determine the enabled state of an **Edit** menu item when a custom dialog item has input focus.

Prototype `#include <CWDropInPanel.h>`

Remarks The IDE sends this request to query the panel about **Edit** menu commands it supports for the dialog item that has the input focus. The panel returns the status of the item specified in the last [reqActivateItem](#) request. The IDE always sends a [reqActivateItem](#) request before sending a [reqFindStatus](#) request.

The IDE automatically supports standard text field items. It only sends the [reqFindStatus](#) request for custom dialog items.

Mac OS On entry:

- `itemHit` contains the value of the Edit menu command that the dialog item does or doesn't support
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

On exit:

- the plug-in sets `itemHit` to true if the item supports the **Edit** command, and false if it doesn't

See Also [“Handling Edit Menu Commands” on page 393](#)

## **reqFirstLoad**

Description	Informs a panel that a new target has been loaded.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends all panels this request when a new target is loaded. Most plug-ins do not need to respond to this request.
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the settings data to operate on</li><li>• <code>toEndian</code> indicates the current endian-ness of the data</li><li>• <code>targetFile</code> specifies the location of the current target file</li></ul>
NOTE	<code>toEndian</code> specifies the current endian-ness of the data, rather than the endian-ness that the plug-in should convert to, in contrast to the <a href="#">reqByteSwapData</a> request.

See Also [“reqFirstLoad Request” on page 355](#)

## **reqGetData**

Description	Asks a panel to extract data values from its panel.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request to ask the panel to copy the current settings of its controls into its settings handle. The IDE sends this request when the user saves the current settings.

Windows A plug-in calls [CWPanleGetCurrentPrefs](#) to obtain its current preferences data handle. Handles can be resized using [CWResizeMemHandle](#). A plug-in extracts values from its panel UI items using calls such as [CWPanleGetItemValue](#) and [CWPanleGetItemText](#), and stores the item values in its current preferences handle directly.

Mac OS On entry:

- `currentPrefs` contains the settings data to operate on.
- `dialog` contains a pointer to the **Target Settings** dialog box.
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box.

On exit:

- `currentPrefs` contains the settings data retrieved from the panel's dialog controls.

See Also [“reqGetData Request” on page 356](#)

## reqGetFactory

Description Asks a panel to return its default settings.

Prototype `#include <CWDropInPanel.h>`

Remarks The IDE asks the panel to return its default settings data.

Windows A plug-in calls [CWPanleGetFactoryPrefs](#) to obtain its factory settings handle. The plug-in then modifies the handle contents to contain default values for all settings.

Mac OS On entry:

- `factoryPrefs` contains the settings data to operate on
- `dialog` contains a pointer to the **Target Settings** dialog box (if `dialog` is not NULL, the dialog box is currently being displayed)
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

On exit:

- `factoryPrefs` contains the panel's default settings data

See Also [“reqGetFactory Request” on page 358](#)

## **reqHandleClick**

Description	Asks a Mac OS panel to act on a mouse click on a custom dialog item.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request when a panel's custom dialog item receives a mouse click. The IDE only sends this request if the item is active.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• event contains the Mac OS event record specifying the mouse click</li><li>• currentPrefs contains a handle to a copy of the panel's current settings data</li><li>• itemHit contains the dialog item that is the target of the mouse click</li><li>• baseItems contains the number of control items that the IDE has already placed in the <b>Target Settings</b> dialog box</li></ul> <p>On exit:</p> <ul style="list-style-type: none"><li>• if the panel sets itemHit to a non-zero value, the IDE will send a <a href="#"><u>reqItemHit</u></a> request to the panel with itemHit containing the item to act upon</li></ul>
See Also	<a href="#"><u>“Handling Keyboard and Mouse Hits” on page 389</u></a>

## **reqHandleKey**

Description	Asks a Mac OS panel to act on a key press directed at a custom dialog item.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request when a panel's custom dialog item receives a keyboard event. The IDE only sends this request if the item is active.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• event contains the Mac OS event record specifying the keyboard event</li></ul>

- `currentPrefs` contains a handle to a copy of the panel's current settings data
- `itemHit` contains the dialog item that is the target of the keyboard event
- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

On exit:

- if the panel sets `itemHit` to a non-zero value, the CodeWarrior IDE will send a [reqItemHit](#) request to the panel with `itemHit` containing the item to act upon

See Also

[“Handling Keyboard and Mouse Hits” on page 389](#)

## reqInitDialog

Description	Asks a panel to prepare to be displayed.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request just before a panel is displayed, when the user selects the panel in the <b>Target Settings</b> dialog. In response to this request, a panel should construct its user interface.
Windows	Panels which include a list box control must construct the contents of the item's list using <a href="#">CWPanelInsertListItem</a> during the <code>reqInitDialog</code> request. The current version of the IDE does not support constructing list boxes from resource descriptions. Panels may also wish to set item values using <a href="#">CWPanelSetItemValue</a> , and enable and disable items using <a href="#">CWPanelEnableItem</a> .
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>dialog</code> contains a pointer to the <b>Target Settings</b> dialog box</li><li>• <code>baseItems</code> contains the number of control items that the IDE has already placed in the <b>Target Settings</b> dialog box</li></ul>
See Also	<a href="#">“reqInitDialog Request” on page 354</a>

## reqInitPanel

Description	Asks a panel plug-in to initialize its state.
Prototype	#include <CWDropInPanel.h>

Remarks	The IDE sends this request when a plug-in is first loaded into memory. In response, plug-ins should initialize any internal state. This request is a deprecated synonym for the <a href="#">reqInitialize</a> request.
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>version</code> contains the runtime version of the panel plug-in API that the IDE supports</li><li>• <code>storage</code> contains the value of a pointer that the IDE saves between requests to the panel until the IDE sends a <a href="#">reqTermPanel</a> request</li><li>• <code>targetfile</code> contains the Mac OS file specification of the active project</li></ul>
See Also	<a href="#">“reqInitialize (reqInitPanel) Request” on page 353</a>

## **reqItemHit**

Description	Informs the panel of user interaction with a dialog item.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request immediately after user interaction with a dialog item. This request is sent in response to mouse clicks and keystrokes directed to the active panel control. In response, plug-ins normally get, set, enable, disable, and show and hide other items.  Before returning to the IDE, and after accounting for any changes in its settings data resulting from the user interaction, a plug-in should indicate whether the current panel settings match the most recently saved settings, and the factory default settings. The IDE uses this information to enable and disable the <b>Revert Panel and Factory Settings</b> buttons.
Windows	Windows panels set the state of the factory flag using <a href="#">CWPanelSetFactoryFlag</a> , and the state of the revert flag using <a href="#">CWPanelSetRevertFlag</a> . plug-ins obtain the current settings handle using <a href="#">CWPanelGetCurrentPrefs</a> , the original (most recently saved) settings using <a href="#">CWPanelGetOriginalPrefs</a> , and the factory default settings using <a href="#">CWPanelGetFactoryPrefs</a> .
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>dialog</code> contains a pointer to the <b>Target Settings</b> dialog box</li></ul>

- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box
- `itemHit` contains the number of the item in the panel that received the event “hit”
- `originalPrefs` contains the settings at the time the **Target Settings** dialog first appeared or the settings panel was first displayed
- `factoryPrefs` contains the settings the panel returned to the IDE for the [reqGetFactory](#)
- `currentPrefs` contains the settings data to operate on

On exit:

- `currentPrefs` contains the changed settings data
- `canRevert` is true if changes in the settings data requires the IDE to activate the **Target Settings** dialog box Revert button
- `canFactory` is true if changes in the settings data requires the IDE to activate the **Target Settings** dialog box Factory button

See Also    [“reqItemHit Request” on page 365](#)

## reqObeyCommand

Description	Asks a Mac OS panel to act on an <b>Edit</b> menu command.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request when the user applies an <b>Edit</b> menu command to a custom panel item having input focus. The request applies to the item specified in the last <a href="#">reqActivateItem</a> request. The IDE always sends <a href="#">reqActivateItem</a> and <a href="#">reqFindStatus</a> requests before sending a <a href="#">reqObeyCommand</a> request.
	The IDE automatically supports <b>Edit</b> menu commands for standard text field items. It only sends the <a href="#">reqFindStatus</a> request for custom dialog items.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>itemHit</code> contains the value of the command to perform. <code>itemHit</code> only specifies one of the commands that the panel supports, based on previous <a href="#">reqFindStatus</a> requests</li></ul>

- `baseItems` contains the number of control items that the IDE has already placed in the **Target Settings** dialog box

On exit:

- if the panel sets `itemHit` to a non-zero value, the CodeWarrior IDE will send a [reqItemHit](#) request to the panel with `itemHit` containing the item to act upon

See Also [“Handling Edit Menu Commands” on page 393](#)

[“Managing Input Focus” on page 373](#)

## reqPutData

Description	Asks a panel to install its settings in its dialog items.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request to indicate that the panel should install values from its settings handle in its panel controls. The IDE sends this request prior to displaying the panel's user interface items.  Panels install values in controls using <a href="#">CWPanelSetItemValue</a> , <a href="#">CWPanelSetItemText</a> , and <a href="#">CWPanelSetItemTextHandle</a> . Panels enable and disable items using <a href="#">CWPanelEnableItem</a> . Panels hide and show items using <a href="#">CWPanelShowItem</a> .
Windows	A panel obtains its current settings handle using <a href="#">CWPanelGetCurrentPrefs</a> .
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>dialog</code> contains a pointer to the <b>Target Settings</b> dialog box</li><li>• <code>baseItems</code> contains the number of control items that the IDE has already placed in the <b>Target Settings</b> dialog box</li><li>• <code>currentPrefs</code> contains the settings data to operate on</li></ul>
See Also	<a href="#">“reqPutData Request” on page 355</a>

## reqReadSettings

Description	Asks a panel to import XML settings data.
Prototype	#include <CWDropInPanel.h>

Remarks	The IDE sends the <code>reqReadSettings</code> request when importing a project from XML. A plug-in should respond by reading all expected name-value pairs from the XML input stream, using calls such as <a href="#">CWReadIntegerSetting</a> , <a href="#">CWGetIntegerValue</a> , <a href="#">CWGetArraySettingElement</a> , <a href="#">CWGetStructureSettingField</a> , and <a href="#">CWGetNamedSetting</a> .
	A panel returns the newly read settings to the IDE by storing them in its current settings handle. The handle may be resized by calling <a href="#">CWResizeMemHandle</a> .
Windows	Windows plug-ins read XML settings into the current settings handle, obtained by calling <a href="#">CWPanlGetCurrentPrefs</a> .
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the settings data to operate on</li></ul>
See Also	<a href="#">“reqReadSettings Request” on page 370</a>

## reqRenameProject

Description	Informs the panel that the project name has changed.
Prototype	<pre>#include &lt;CWDropInPanel.h&gt;</pre>
Remarks	The IDE asks the panel to change its settings information based on a new project name. The IDE sends this request when creating a new project based on a project stationery document. The IDE does <i>not</i> send this request when the user renames a project using Windows Explorer, or the Mac OS Finder.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>oldtargfile</code> contains the Mac OS file specification for the old target file</li><li>• <code>currentPrefs</code> contains the settings data to operate on</li></ul> <p>On exit:</p> <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the changed settings data</li></ul>
See Also	<a href="#">“reqRenameProject Request” on page 365</a>

## reqSetupDebug

Description	Informs the panel that debugging has been enabled or disabled.
-------------	--

## Settings Panel Plug-in API Reference

### *reqTermDialog*

---

Prototype	#include <CWDropInPanel.h>
Remarks	The IDE asks the panel to change its settings information to support debugging of the current project. If the state of debugging does not affect a panel, this request can be ignored.
Windows	Windows plug-ins can determine whether debugging is enabled by calling <a href="#">CWPanelGetDebugFlag</a> .
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the settings data to operate on</li><li>• <code>debugOn</code> is true if debugging the project is enabled, false if debugging is disabled</li></ul> <p>On exit:</p> <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the changed settings data</li></ul>
See Also	<a href="#">“reqSetupDebug Request” on page 364</a> <a href="#">“CWPanelGetDebugFlag” on page 424</a>

## reqTermDialog

Description	Informs a panel that its user interface is about to be closed.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE asks the panel to prepare to be removed from the <b>Target Settings</b> dialog box. The IDE sends this request when the user changes panels or when the user closes the <b>Target Settings</b> dialog box.
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>dialog</code> contains a pointer to the <b>Target Settings</b> dialog box</li><li>• <code>baseItems</code> contains the number of control items that the IDE has already placed in the <b>Target Settings</b> dialog box</li></ul>
See Also	<a href="#">“reqTermDialog Request” on page 355</a>

## reqTermPanel

Description	Informs the panel that it is about to be unloaded from memory.
Prototype	#include <CWDropInPanel.h>

Remarks	<p>The IDE issues this request prior to unloading a panel from memory. This constant is a deprecated synonym for the <a href="#">reqTerminate</a> request.</p> <p>plug-ins should clean up internal state, and free any resources acquired when responding to the <a href="#">reqInitPanel</a> request.</p>
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"> <li>• <b>version</b> contains the runtime version of the panel plug-in API that the IDE supports</li> <li>• <b>storage</b> contains the value of a pointer that the IDE saved between requests to the panel since the IDE sent a <a href="#">reqInitPanel</a> request</li> <li>• <b>targetfile</b> contains the Mac OS file specification of the active project</li> </ul>
See Also	<p><a href="#">“reqTerminate (reqTermPanel) Request” on page 353</a></p>

## reqUpdatePref

Description	Asks a panel to update the version of its settings data.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE asks the panel to update an older version of its settings data to the latest version. plug-ins modify the contents of the current settings handle to return the updated settings to the IDE.
<b>NOTE</b>	Plug-ins must always store the data version in the first 16-bit integer of the settings data handle.

---

Typically, plug-ins construct a new settings handle containing default values for current settings, and then copy and convert any values still supported from the old settings data. The current default settings can be constructed using the same code that responds to the [reqGetFactory](#) request. The current settings handle may need to be expanded, by calling [CWResizeMemHandle](#).

Windows	Windows plug-ins obtain the current settings data handle by calling <a href="#">CWPanelGetCurrentPrefs</a> . plug-ins return update settings by modifying the handle's contents directly.
Mac OS	On entry:

---

- `currentPrefs` contains the settings data to operate on

On exit:

- `currentPrefs` contains the changed settings data

See Also [“reqUpdatePref Request” on page 362](#)

## reqValidate

Description	Asks a plug-in to determine any actions required by the IDE in response to changes in the plug-in's settings.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request to determine whether changes in the panel's settings (the differences between the original state and the current state) require the project to reset its file paths, to be recompiled, or to be relinked.
Windows	Windows plug-ins obtain the current settings by calling <a href="#">CWPanelGetCurrentPrefs</a> and the most recently saved settings by calling <a href="#">CWPanelGetOriginalPrefs</a> .  To indicate whether project paths should be reset, a plug-in calls <a href="#">CWPanelSetResetPathsFlag</a> . To indicate whether the project should be recompiled, a plug-in calls <a href="#">CWPanelSetRecompileFlag</a> . To indicate whether the project should be relinked, a plug-in calls <a href="#">CWPanelSetRelinkFlag</a> . To indicate whether the browse information for the current project should be reparsed, a plug-in calls <a href="#">CWPanelSetReparseFlag</a> .
Mac OS	<p>On entry:</p> <ul style="list-style-type: none"><li>• <code>originalPrefs</code> contains the most recently saved settings data</li><li>• <code>currentPrefs</code> contains the current settings data</li></ul> <p>On exit:</p> <ul style="list-style-type: none"><li>• <code>reset</code> is true if the project's file paths should be reset.</li><li>• <code>recompile</code> is true if the project should be recompiled.</li><li>• <code>relink</code> is true if the project should be relinked.</li><li>• <code>reparses</code> is true if the project's files should be reprocessed by the class browser (this flag is currently unused by the IDE)</li></ul>

See Also    [“reqValidate Request” on page 361](#)

## reqWriteSettings

Description	Asks a plug-in to export its settings data as XML.
Prototype	#include <CWDropInPanel.h>
Remarks	The IDE sends this request to tell a plug-in to export its settings as XML. In response, a plug-in should call  The IDE sends the <code>reqWriteSettings</code> request when exporting a project to XML. A plug-in should respond by writing its settings data as name-value pairs to the XML output stream, using calls such as <a href="#">CWWriteIntegerSetting</a> , <a href="#">CWSetIntegerValue</a> , <a href="#">CWGetArraySettingElement</a> , <a href="#">CWGetStructureSettingField</a> , and <a href="#">CWGetNamedSetting</a> .
Windows	Windows panels obtain their current settings handle to write as XML by calling <a href="#">CWPanleGetCurrentPrefs</a> .
Mac OS	On entry: <ul style="list-style-type: none"><li>• <code>currentPrefs</code> contains the current settings data</li></ul>
See Also	<a href="#">“reqWriteSettings Request” on page 370</a>

## supportsByteSwapping

Description	A panel flag indicating that a plug-in knows how to swap the byte ordering of its settings data.
Prototype	#include <CWDropInPanel.h>
Remarks	A panel sets this flag in its <a href="#">PanelFlags</a> to indicate that it supports the <a href="#">reqByteSwapData</a> request. Panels should support this request if at all possible.
See Also	<a href="#">“PanelFlags” on page 483</a> <a href="#">“reqByteSwapData” on page 500</a> <a href="#">“reqByteSwapData Request” on page 359</a> <a href="#">“PanelFlags Structure” on page 340</a>

## supportsTextSettings

Description	A panel flag indicating that a plug-in supports XML import and export of its settings data.
Prototype	#include <CWDropInPanel.h>
Remarks	A panel sets this flag in its <a href="#">PanelFlags</a> to indicate that it supports the <a href="#">reqReadSettings</a> and <a href="#">reqWriteSettings</a> requests, used when importing and exporting XML data, respectively.
	In response to these requests, a panel should read and write a series of XML settings corresponding to its settings data. Panels should support these requests if at all possible.
See Also	<a href="#">“Importing and Exporting Settings Data” on page 366</a> <a href="#">“PanelFlags” on page 483</a>

## usesStrictAPI

Description	A panel flag indicating that a Mac OS plug-in uses exclusively the CodeWarrior Plug-in API to manipulate user interface controls.
Prototype	#include <CWDropInPanel.h>
Remarks	Panel plug-ins set this bit in the <code>dropinflags</code> field of their <a href="#">DropinFlags Structure</a> to indicate that they utilize only plug-in API routines for manipulating user interface controls, rather than Mac OS dialog manager routines. This allows the IDE to construct a panel’s user interface with more flexibility, without necessarily creating or exposing a Mac OS dialog manager dialog window.
	All new Mac OS panel plug-ins should set this flag, and should also enable <a href="#">CW_STRICT_DIALOGS</a> at compile time, to help enforce dialog access using the API only.
See Also	<a href="#">“CW_STRICT_DIALOGS” on page 494</a> <a href="#">“PanelFlags” on page 483</a>

# Settings Panel Result Codes

The IDE supports returning additional error codes from panel plug-ins. This section documents the panel-specific error codes.

The API defines the following settings panel error codes:

- [kBadPrefVersion](#)
- [kMissingPrefErr](#)
- [kSettingNotFoundErr](#)
- [kSettingTypeMismatchErr](#)
- [kInvalidCallbackErr](#)
- [kSettingOutOfRangeErr](#)

## kBadPrefVersion

Description	Indicates that a panel cannot convert its preference data to the requested version.
Definition	#include <CWDropInPanel.h> /* Windows */ #include <CWDropInPanel.h> /* Mac OS */
Remarks	A plug-in returns this error code to the IDE when they receive a request to convert a version of the settings data that they do not understand to the current version. This happens, for example, when a project containing a newer version of the plug-in data is opened by an IDE running an older version of a plug-in. It also happens when the IDE asks a plug-in to an old version of its data that it may no longer support.

## kMissingPrefErr

Description	A currently unused error code.
Definition	#include <CWDropInPanel.h> /* Windows */ #include <CWDropInPanel.h> /* Mac OS */

## kSettingNotFoundErr

Description	Indicates that an XML setting by the specified name could not be found.
-------------	---

Definition    `#include <CWDropInPanel.h> /* Windows */  
              #include <CWDropInPanel.h>                  /* Mac OS */`

Remarks    The IDE returns this error code when a plug-in calls either [CWGetNamedSetting](#) or one of the CWRead calls, to obtain a setting which does not appear in the plug-in's input XML stream.

## **kSettingTypeMismatchErr**

Description    Indicates that the requested setting was not of the expected type.

Definition    `#include <CWDropInPanel.h> /* Windows */  
              #include <CWDropInPanel.h>                  /* Mac OS */`

Remarks    The IDE returns this error code when a panel plug-in attempts to read an XML settings, and a setting by the requested name is available, but the type of the setting does not match that implied by the API call.

For example, the IDE returns this result if a plug-in attempts to read a string settings using [CWGetIntegerValue](#).

## **kInvalidCallbackErr**

Description    Indicates that the called routine is not available during the current request.

Definition    `#include <CWDropInPanel.h> /* Windows */  
              #include <CWDropInPanel.h>                  /* Mac OS */`

Remarks    The IDE returns this error code when a panel plug-in calls an API routine that is not supported for the current request. For example, the IDE returns this value when plug-ins call one of the CWRead or CWWrite routines, or one of the other XML import or export routines, during a request other than [reqReadSettings](#) or [reqWriteSettings](#).

## **kSettingOutOfRangeErr**

Description    An internally used error code.

Definition    `#include <CWDropInPanel.h> /* Windows */  
              #include <CWDropInPanel.h>                  /* Mac OS */`

Remarks    The does not currently return this error code to panel plug-ins. This may change in the future.



# Version Control System Plug-in API

---

The CodeWarrior IDE supports version control system (VCS) plug-ins. You use version control systems to keep track of the differing versions of your source code files as you modify them. This allows you to check files in to a repository that others can share, making change tracking easier, especially in software development teams. This chapter explains the CodeWarrior IDE Plug-in API with respect to VCS plug-in implementation issues.

## What's New in the VCS API?

The current VCS API is version 9.

The following changes were introduced with the preceding version of the API (version 8):

- The "date" field of the `CWVCSVersionData` was changed from being a C standard library `tm` structure to being a [`CWFfileTime`](#) structure. The latter is language neutral, and better complies with host OS conventions for time and date representation.
- Replaced the [`CWFfileStateChanged`](#) callback with a new [`CWVCSStateChanged`](#) callback, which is identical except for passing the `version` parameter by reference, again for reasons of language neutrality.
- The `CWVCSItem` struct now includes the checkout state of the item. This field should be set by a plug-in before calling [`CWSetVCSItem`](#).

The following changes were introduced with the preceding version of the API (version 7):

- native Windows support for DOS-style paths

- progress dialog feedback during long operations
- ability to change the menu item names and command descriptions
- Pre/PostFileActions can be used to prepare files for processing
- plug-ins will receive idle events for processing
- notification of when the plug-in's preferences or main VCS preferences change
- ability to invoke the IDE's visual difference for comparing files
- memory allocation routines

---

**NOTE** This chapter has been only minimally updated to reflect version 8 API changes (a thorough revision is imminent). Check the Metrowerks web site (<http://www.metrowerks.com>) for documentation and SDK updates.

---

## Overview

The CodeWarrior IDE provides support for external version control systems through a plug-in and preference panel interface.

Beginning with version 7 of the VCS API, version control plug-ins and preference panels are implemented similarly to other IDE plug-ins such as compilers and linkers. Routines which are accessible to general plug-ins, including memory allocation, obtaining of preference information, and file handling functions are now accessible to VCS plug-ins.

In order to use a VCS from within the IDE, the Version Control preferences panel family's General panel must have a version control system selected. When this is done the IDE will dispatch calls to the VCS library, if one is present. A description of the user interface modifications to the IDE are in the CodeWarrior IDE Version Control System User Interface Specification.

All communication between the IDE and the plug-in is done through several entry points. The standard entry points for plug-ins which provide the plug-in's name, capability flags, display name, and associated preference panels are supported along with the resource based methods of specifying plug-in information.

The VCS plug-in is a specific part of the plug-in architecture. In addition to the standard callback mechanisms available for generic plug-ins, there are several callbacks and communication methods that are unique to VCS plug-ins. These provide the additional functionality for VCS systems to interact with the IDE.

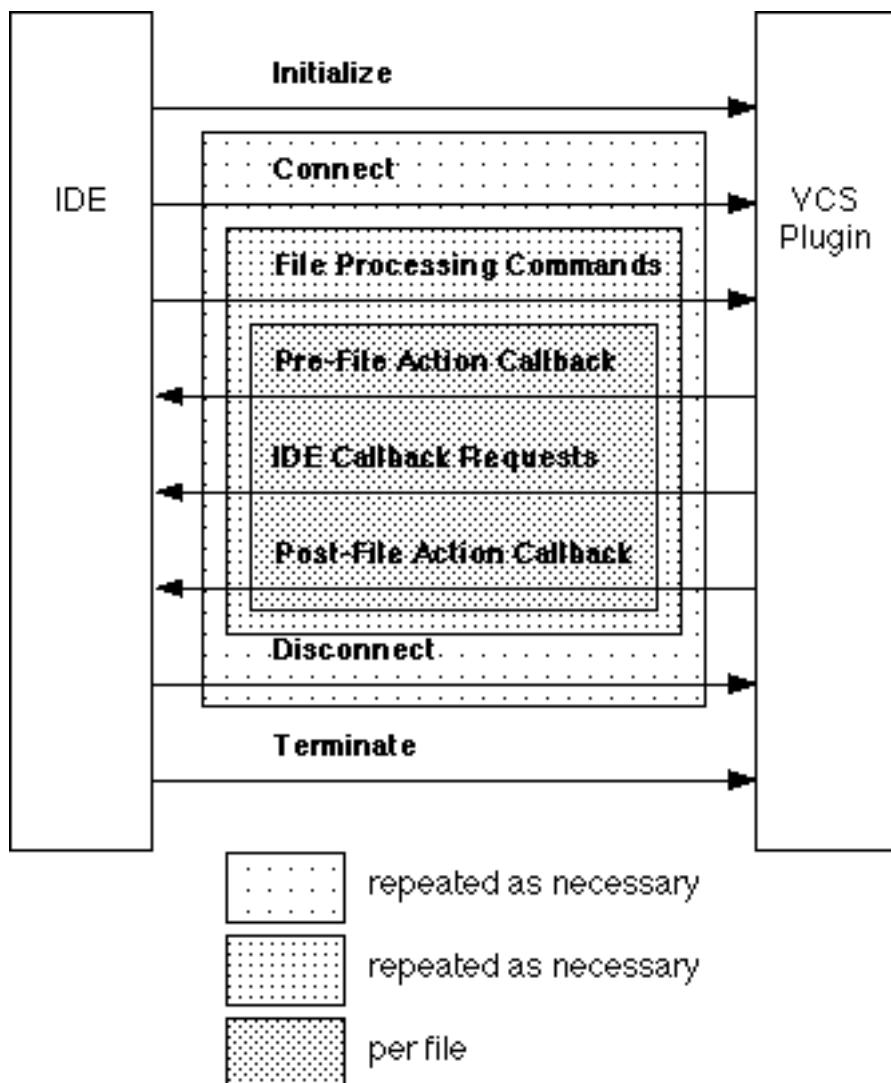
This chapter covers the following topics:

- [Operational Phases](#)
- [VCS Plug-in Interface](#)
- [Plug-in Callbacks](#)
- [VCS Requests](#)
- [VCS API Version 1 Compatibility](#)

## Operational Phases

There are six (6) phases of operation for the plug-in used for version control. These are initialization and termination, command support negotiation, database connection and disconnection, file processing, information, and messaging. These are illustrated in [Figure 12.1](#).

**Figure 12.1 Operational Phases**



### 1. Initialization / Termination

When the plug-in is loaded or unloaded, (typically at startup and shutdown of the IDE), an initialization or termination command is issued. These commands are provided to the VCS in order for it to handle the startup/shutdown housekeeping tasks which it may require.

## **2. Command Support Negotiation**

The IDE and VCS plug-in negotiate which commands are supported.

## **3. Database Connection / Disconnection**

Commands are provided to connect/disconnect the VCS to/from its database.

## **4. File processing**

Commands are provided for general VCS database file operations.

## **5. Information**

At various times, the IDE will notify the VCS plug-in that the state of a file or the VCS plug-in's preference panel have changed. Additionally, the IDE may allow time for the plug-in to perform periodic actions.

## **6. Messaging**

Callbacks are provided which allow the VCS to handle the displaying of dialogues when user input is required and to handle the output of information to the user via the IDE's message window.

# **VCS Plug-in Interface**

## **The Main Entry Point**

A VCS program communicates to the IDE through a series of entry points. The VCS architecture uses the standard entry points of the IDE plug-in library. Aside from the main entry point, all of them are of the type CWPLUGIN\_ENTRY.

A plug-in's `main()` entry point is used by the IDE to deliver commands to the VCS plug-in.

## Version Control System Plug-in API

### The Main Entry Point

---

```
pascal short main (CWPluginContext context)
```

where tVCSEntryPoint is declared as

```
pascal short
```

on the Macintosh and as

```
__declspec(dllexport) F_PASCAL(short)
```

under Windows.

This routine returns a short. This value must be the same as the command status, that is, a coarse value indicating the success or failure of the operation the IDE requested. An item-specific status is contained within the CWVCSItems that the IDE passes to the plug-in to operate upon.

A Version 7 or later VCS plug-in main routine must have a specific structure which is different from the structure of a standard plug-in callback. This structure allows version 7 and later plug-ins to be backwards compatible with versions of the IDE prior to the Pro 4 release (although only specific callbacks and requests are supported—see the section on V1 Compatibility for details).

Immediately upon entry the VCS must call [CWAllowV1Compatibility](#) and check its return value. If it returns an error aside from cwNoErr, the error must be immediately returned by the plug-in and no attempt should be made to process the request. This allows the plug-in to either run in version 1 compatibility mode or disallow an incompatible IDE from loading it.

After passing through the V1 compatibility check, the plug-in obtains the request from the IDE by using the standard callback [CWGetPluginRequest](#). The value obtained from this callback is then used to determine why the plug-in is being called by the IDE.

The return value of the call to the plug-in must be returned in a three step process. The plug-in returns a value indicating success or failure by first calling [CWSetCommandStatus](#) with one of the status constants indicating relative completion of the task. If the command status is set outside of the main function, then it should be obtained with [CWGetCommandStatus](#) for the other two steps.

The plug-in then calls [CWDonePluginRequest](#) with this same command status value. Finally, the plug-in returns, again with the same value.

The generic structure of a main entry point is presented in [Listing 12.1](#). This example illustrates a plug-in that runs in V1 compatibility mode.

### **Listing 12.1 VCS plug-in main() entry point**

---

```
pascal short main(CWPluginContext context)
{
    Boolean isInV1Mode;
    CWResult theErr=CWAllowV1Compatibility(context, true,
                                             &isInV1Mode);

    if(theErr!=cwNoErr)
        return(theErr);
    if(isInV1Mode)
    {
        // perform any special processing to
        // implement unsupported features
    }
    // set status in case of error
    theErr=CWSetCommandStatus(context, cwCommandStatusUnknown);
    ASSERT(theErr==cwNoErr); // or other error handling code
    long request;
    theErr=CWGetPluginRequest(context, &request);
    ASSERT(theErr==cwNoErr); // or other error handling code
    switch(request)
    {
        case reqInitialize:
        ...
    }

    CWVCSCommandStatus endResult;
    theErr=CWGetCommandStatus(context, &endResult);
    ASSERT(theErr==cwNoErr);
    endResult=CWDonePluginRequest(context, endResult);

    return(endResult);
}
```

---

The main entry point should be stored as the main entry point of the main code fragment (Mac CFM/PPC) or as the entry point of the DLL (Windows). It must be exported along with all other plug-in entry points. See the CW IDE plug-in Developer's Guide for more information.

## **VCS Plug-in Structure**

A VCS plug-in can have many possible structures for handling requests. This section will introduce some possible solutions for handling command negotiation, structure for handling variants of commands, and working with the file lists. These samples make use of callbacks and structures described in the plug-in Callbacks section.

### **Handling Negotiation Requests**

The sample plug-ins included with the plug-in SDK make use of two different architectures, centering around how the command negotiation is handled. One solution is to search for the negotiation request variant in the main entry point before dispatching the request to the switch statement:

---

#### **Listing 12.2 Separate negotiation and regular request handling**

---

```
pascal short main(CWPluginContext context)
{
    ... initialization code ...
    long request;
    theErr=CWGetPluginRequest(context, &request);
    ASSERT(theErr==noErr); // or other error handling code
    Boolean isNegotiationRequest;
    theErr=CWIsCommandSupportedRequest(context,
                                         &isNegotiationRequest);
    ASSERT(theErr==noErr); // or other error handling code
    if(isNegotiationRequest)
    {
        // request is to see if an action is supported by the plug-in
        CWVCSCommandStatus isSupported=
            cwCommandStatusCommandUnsupported;
        ... determine request variant (advanced/recursive) ...

        switch(request)
```

```

{
    case ...:
        <<if command is supported>>
        isSupported=cwCommandStatusCommandSupported;
        ...
    }
    theErr=CWSetCommandStatus(context, isSupported);
}
else
{
    // request is for an action to be taken
    switch(request)
    {
        ... dispatch request to regular handling functions ...
    }
}
... cleanup code ...
}

```

---

The other example structure is to move the handling of negotiation requests into the regular request handling functions themselves. In that case, the main function's structure would include just the second switch statement (as in the previous Main Entry Point structure section), but the handling functions would have the additional if statement, as illustrated by this possible handler function called in response to [reqFileAdd](#) requests:

### **Listing 12.3 Combined negotiation and regular request handling**

```

int HandleAddFileRequest(CWPluginContext context)
{
    Boolean isNegotiationRequest=false;
    OSerr theErr=CWIIsCommandSupportedRequest(context,
                                                &isNegotiationRequest);
    ASSERT(theErr==noErr); // or other error handling code
    if(isNegotiationRequest)
    {
        ... processing to determine if command is supported ...
    }
    else
    {
        ... processing to execute the add file request ...
    }
}

```

---

```
}
```

Either will work fine, although the first has the benefit of isolating out the handling of negotiation requests and may be helpful in debugging.

### **Handling Variants of Commands**

The IDE sends gross level values as its requests to the plug-in. There is a finer granularity to the requests than the `req*` request constants, however. The requests can have any of the following three extra properties that can be used in combination:

- **negotiation request**—the IDE is trying to see if the command is supported by the plug-in. You can determine if the request is a negotiation request by using the [`CWIsCommandSupportedRequest`](#) callback.
- **advanced request**—the plug-in should ask the user for extra information with regards to the action. For example, on a status request the regular command may return simply the status of the file as far as the user is concerned. If the request is an advanced request, however, the plug-in may choose to allow the user to display the file status for all users working on the file. You can determine if the request is an advanced request by using the [`CWIsAdvancedRequest`](#) callback.
- **recursive request**—the plug-in should process directories and all of its children recursively contained in the list of files and directories passed to the plug-in by the IDE instead of operating on the directory itself. You can determine if the request is a recursive request by using the [`CWIsRecursiveRequest`](#) callback.

These different variants of requests can be mixed together. For example, the plug-in can receive a [`reqFileAdd`](#) for which [`CWIsCommandSupportedRequest`](#) and [`CWIsRecursiveRequest`](#) return true. In this case, the IDE would be asking the plug-in if it allows the user to add whole directory structures to the database, passed by the IDE to the plug-in as only a single [`CWFfileSpec`](#) for the parent directory.

In general, you will need to determine which variant of the command is being passed inside of your handling functions. One

way of properly dealing with all of these different variants is illustrated in [Listing 12.4](#), assuming that the handling function is also doing the command negotiation processing.

#### **Listing 12.4    Command handling example**

```
int HandleFileAddRequest(CWPluginContext context)
{
    Boolean isNegotiation, isAdvanced, isRecursive;
    OSerr theErr=CWIIsCommandSupportedRequest(context,
                                                &isNegotiation);
    ASSERT(theErr==noErr); // or other error handling code
    theErr=CWIIsAdvancedRequest(context, &isAdvanced);
    ASSERT(theErr==noErr); // or other error handling code

    theErr=CWIIsRecursiveRequest(context, &isRecursive);
    ASSERT(theErr==noErr); // or other error handling code
    if(isNegotiation)
    {
        ... check if a command variant is supported ...
        if(isAdvanced && isRecursive)
            ...
        else if(isAdvanced)
            ...
        else if(isRecursive)
            ...
        else
            ... standard functionality case ...
    }
    else
    {
        ... execute a command variant ...
        if(isAdvanced && isRecursive)
            ...
        else if(isAdvanced)
            ...
        else if(isRecursive)
            ...
        else
            ... standard functionality case ...
    }
}
```

```
}
```

---

## Working with File Lists

The IDE passes in a list of files and directories for the plug-in to process. The plug-in must use accessor routines to get at this list of files. As there are memory freeing requirements, the exact structure of how to accomplish this iteration is illustrated below.

First, a plug-in would use [CWGetVCSItemCount](#) to return the number of files and directories passed to it by the IDE. Then, the plug-in loops over this variable to process each individual item. An item is retrieved with [CWGetVCSItem](#). It is then processed according to the type of request the IDE has made. As each file is processed, the plug-in calls [CWPreFileAction](#) before beginning to process an individual file, then [CWFfileStateChanged](#) if the plug-in either modified the file on disk or its checkout state in the database has changed, and then finally calls [CWPostFileAction](#) to send notification of the completion of the file processing. After processing, the eItemStatus field of the VCS item is set to an appropriate value indicating whether the processing of that item was completed successfully or not. Finally, the plug-in then communicates this information back to the IDE along with any other changed information for the item by calling [CWSetVCSItem](#).

This is illustrated in [Listing 12.5](#). This code assumes that the items all correspond to files and does not deal with recursive operations.

### **Listing 12.5 Basic file list processing**

---

```
long numItems;
OSErr theErr=CWGetVCSItemCount(context, &numItems);
ASSERT(theErr==noErr); // or other error handling
for(long I=0; i<numItems; I++)
{
    CWVCSItem theItem;
    theErr=CWGetVCSItem(context, I, &theItem);
    ASSERT(theErr==noErr); // or other error handling
    // +++ here we assume for simplicity that the item is
    // a file. We should check to see if it is a directory
    // for recursive processing.
```

---

```
theErr=CWPreFileAction(context, &theItem.fsItem);
ASSERT(theErr==noErr); // or other error handling
... file processing code ...
if(... file state changed ...)
{
    theErr=CWFFileStateChanged(context, &theItem.fsItem,
    ... new checkout state ...,
    theItem->version);
    ASSERT(theErr==noErr); // or other error handling
}
theItem.eItemStatus=
(... operation successful ...) ? cwItemStatusSucceeded :
cwItemStatusFailed;
theErr=CWSetVCSItem(context, I, &theItem);
ASSERT(theErr==noErr); // or other error handling
}
```

---

Note that you must pass [CWGetVCSItem](#) a pointer to a memory location holding enough room for a CWVCSItem.

### Other Plug-in Entry Points

The VCS plug-in developer must specify further information about the plug-in and its capabilities to allow the IDE to call it properly. This is achieved the same way as for standard plug-ins:

- Internal name and display name – [“Specifying a Plug-in's Dropin and Display Names” on page 70](#)
- Capability flags – See [“Specifying Plug-in Capabilities” on page 65](#)
- Associated preference panels – [“Specifying Associated Settings Panels” on page 72](#)
- Panel families – [“Specifying Panel Family” on page 73](#)

Although providing some of these entry points is optional (see [“Informational Entry Points” on page 65](#)), it is recommended that you support at least the internal and display names, and capability flags.

On the Macintosh, this information may be supplied through use of the resources (see [“Mac OS Plug-in Resource Formats” on page 641](#))

although use of the entry points is recommended to increase portability to other platforms.

When programming native Windows plug-ins, only the entry points are supported.

### **VCS Plug-in Capability Flags**

A VCS plug-in must specify its capability flags using either the ['Flag' Resource](#) for Macintosh plug-ins, or through the [CWPlugin\\_GetDropInFlags Entry Point](#). The capability flags have a new format for Version 7 and later plug-ins.

First, the `rsrcversion` must be set to 3. If it is not, the Pro 4 and later IDEs will believe that the VCS plug-in is a version 1 plug-in and the plug-in will either refuse to load if it is not V1 compatible or will be forced to run in V1 compatibility mode.

The `earliestCompatibleAPIVersion` should be set to either 1 if your VCS plug-in can run in V1 compatibility mode, or 7 if your plug-in cannot run in the compatibility mode.

The `dropinflags` field is a bitfield corresponding to special features of the VCS API that the plug-in can turn on and off. This field must be the bitwise or of the following constants:

```
enum // dropinflags
{
    vcsDoesntPrePostSingleFile,
    vcsDoesntPrePostRecursive,
    vcsRequiresEvents,
    vcsWantsIdle,
    vcsSupportsPrefsChanged
};
```

These capabilities are as follows:

#### **vcsDoesntPrePostSingleFile**

Versions 7 and later of the VCS API provide a set of callback functions, [CWPreFileAction](#) and [CWPostFileAction](#) designed to be called before and after a file is processed by a VCS plug-in. These callbacks tell the IDE to make a file available for reading and writing.

Under version 1, this was done automatically for all files that were passed to the plug-in before the IDE sent the request. In Version 7, however, if the plug-in can call these functions before and after processing a file it is highly recommended to allow for future compatibility.

By setting the `vcsDoesntPrePostSingleFile` capability flag, you are telling the IDE that you would like it to prepare the file for any calls affecting only a single file before it calls the main entry point. In other words, you are telling the IDE that you cannot use the file action routines due to a lack of information and want to be guaranteed that the file will be accessible.

If you intend to run in V1 compatibility mode, set this flag to true to have the same behavior expected under both environments: V1 where the file action callbacks are not implemented and cannot be called, and V7 where they are allowed.

#### **vcsDoesntPrePostRecursive**

By setting this flag to true, you are telling the IDE that when your plug-in is called to perform a recursive operation, you want it to prepare every file beforehand to insure accessibility rather than calling the file action routines yourself for every specific file. If you use the action routines before and after processing each individual file instead of setting this flag, in future versions of the IDE the programmer will still be allowed to work on files involved in a recursive VCS operation while they are not being used. With the flag set, however, users will need to wait for the operation to finish before being able to work with the files involved in the operation.

If you intend to run in V1 compatibility mode, set this flag to true to have the same behavior expected under both environments: V1 where the file action callbacks are not implemented and cannot be called, and V7 where they are allowed.

#### **vcsRequiresEvents (Macintosh)**

If you specify this flag on, you will be telling the IDE that your plug-in does special event processing that makes it require it to receive events instead of the IDE. This flag would need to be set, for example, if your plug-in uses AppleEvents to communicate with

another program. If it is off, the IDE will attempt to respond to all events instead of passing them to the plug-in.

#### **vcsWantIdle (Version 7 and later only)**

If you set this flag to true, when running in Version 7 mode (Pro 4 IDE and later) your plug-in will receive [reqIdle](#) requests from the IDE periodically to allow for idle time tasks/keepalive tasks to be executed. If this flag is not set, your plug-in will not receive idle requests.

Since idle requests are not sent by pre-Pro 4 IDEs, if you plan to allow your plug-in to run in V1 compatibility mode you should set this flag to off as idle events are not sent to plug-ins in V1 compatibility mode. If you require the equivalent of idle time processing and need to run in V1 compatibility mode you must implement your own solution, for example, through VBL tasks or thread messaging.

#### **vcsSupportsPrefsChanged**

A plug-in sets this flag to indicate that it wants to be notified when the project's preferences change. When this flag is set, the IDE will send a [reqPrefsChange](#) request when the settings for a project change.

Note that the IDE currently makes this request in a way that ensures the plug-in will *not* be called reentrantly. That is, there currently is no risk of a VCS plug-in being sent the [reqPrefsChange](#) request while another request is being serviced.

## **Plug-in Callbacks**

Functionality to obtain information from the IDE as well as communicate information back, perform actions, and inquire into file states is accomplished using callbacks. There are both standard callbacks as well as callbacks which are specific to VCS plug-ins.

Not all of these callbacks are available when running in V1 compatibility mode. See the later section on V1 compatibility mode for which callbacks are available to be used. If you try to use a callback in an inappropriate mode, if you pass in an invalid

parameter, or if it fails, it will return an error of type [CWResult](#), so be sure to check the return error values of the callbacks in your code to ensure proper operation.

All files are specified as a [CWFileSpec](#). On Macintosh systems, this will be a standard FSSpec, and on Windows systems it is a structure with one member variable, path, which will contain the full path to the file or directory:

## Standard Callbacks

The following standard callbacks are accessible to VCS plug-ins:

Callback	1	2	3
<a href="#">CWGetPluginRequest</a>	X		X
<a href="#">CWDonePluginRequest</a>	X		X
<a href="#">CWGetAPIVersion</a>	X		X
<a href="#">CWGetIDEInfo</a>	X		
<a href="#">CWGetCallbackOSError</a>		X	
<a href="#">CWSetPluginOSError</a>			
<a href="#">CWGetFileText</a>			
<a href="#">CWReleaseFileText</a>			
<a href="#">CWReportMessage</a>			X
<a href="#">CWAAlert</a>			X
<a href="#">CWShowStatus</a>			X
<a href="#">CWUserBreak</a>			X
<a href="#">CWGetNamedPreferences</a>			X
<a href="#">CWStorePluginData</a>			
<a href="#">CWGetPluginData</a>			
<a href="#">CWCreateNewTextDocument</a>			X
<a href="#">CWAAllocateMemory</a>		X	

## Version Control System Plug-in API

### Standard Callbacks

---

Callback	1	2	3
<a href="#">CWFreeMemory</a>	X		
<a href="#">CWAllocMemHandle</a>	X		X
<a href="#">CWFreeMemHandle</a>	X		X
<a href="#">CWGetMemHandleSize</a>	X		X
<a href="#">CWResizeMemHandle</a>	X		X
<a href="#">CWLockMemHandle</a>	X		X
<a href="#">CWUnlockMemHandle</a>	X		X
<a href="#">CWPreDialog</a>	X	X	X
<a href="#">CWPostDialog</a>	X	X	X
<a href="#">CWPreFileAction</a>			X
<a href="#">CWPostFileAction</a>			X

1: callback can be used during [reqInitialize](#) and [reqTerminate](#) under Pro 4 IDE.

2: callback is new with plug-in API Version 9 (Pro 4 IDE)

3: callback is supported in V1 compatibility mode

These standard callbacks are documented in the CodeWarrior IDE Plug-in Developer's Guide. The following callbacks are either new to the plug-in API or have a slightly different behavior for VCS plug-ins than others as a result of the combination of the VCS API with the plug-in API:

### **CWGetPluginData/CWStorePluginData**

These two functions can be used by a VCS plug-in to store non-volatile information associated with a project. Each project has its unique data storage area. These functions have behavioral differences for VCS plug-ins in that the data is stored projectwide and not on a per-target basis. The filenumber passed in as an argument should always be -1. Any tag that is not all lowercase letters can be used.

### **CWReportMessage**

This function will display a message inside of the VCS messages window instead of the errors & warnings window. Any file and line information that is passed in with the message is ignored for VCS plug-ins.

### **CWPreDialog**

```
CW_CALLBACK CWPreDialog(CWPluginContext context);
```

This function takes the `context` that is passed in to the main entry point as its only argument. It tells the IDE to prepare for the plug-in to display a dialog. By calling this function, the IDE can disable any windows and make preparations for the Dialog Manager to be called so it can function properly.

### **CWPostDialog**

```
CW_CALLBACK CWPostDialog(CWPluginContext context);
```

This function takes as its argument the `context` passed into the main entry point. It is used by the plug-in to notify the IDE that the plug-in's dialog is finished displaying and has been destroyed so the IDE can reset the windows for the user.

### **CWPreFileAction**

```
CW_CALLBACK CWPreFileAction  
(CWPluginContext context,  
 const CWFfileSpec *theFile);
```

This callback takes as its arguments the `context` passed to the main entry point along with a [CWFfileSpec](#) that contains the appropriate file system specification for a file.

The plug-in can call this function before it attempts to do any type of processing of a file. By calling [CWPreFileAction](#), the IDE will make sure that it can do everything possible to insure accessibility of the file by, for example, closing down any access paths the IDE has to the file so the plug-in can open it for exclusive read/writes.

### **CWPostFileAction**

```
CW_CALLBACK CWPostFileAction  
(CWPluginContext context,  
 const CWFfileSpec *theFile);
```

Plug-ins that process files should use this callback after all processing of a file is done and all of the plug-in's access paths to the file have been closed. This then allows the IDE to reestablish any access it needs to the file in order to function. The callback takes the context that is passed to a plug-in's main entry point along with a [CWFileSpec](#) indicating which file has just finished being processed by the plug-in.

### VCS Specific Callbacks

The VCS extension of the base plug-in API features the following VCS specific callbacks used to initiate actions, retrieve information from the IDE, and send information back to the IDE. These functions are:

Callback	1	2
<a href="#">CWAllowV1Compatibility</a>		
<a href="#">CWGetComment</a>		
<a href="#">CWFileStateChanged</a>		
<a href="#">CWVCSStateChanged</a>		
<a href="#">CWGetProjectFileSpecifier</a>		
<a href="#">CWIsAdvancedRequest</a>		
<a href="#">CWIsRecursiveRequest</a>		
<a href="#">CWIsCommandSupportedRequest</a>		
<a href="#">CWGetDatabaseConnectionInfo</a>		
<a href="#">CWGetCommandStatus</a>		
<a href="#">CWSetCommandStatus</a>		
<a href="#">CWGetVCSItemCount</a>		
<a href="#">CWGetVCSItem</a>		
<a href="#">CWSetVCSItem</a>		
<a href="#">CWGetVCSPointerStorage</a>		
<a href="#">CWSetVCSPointerStorage</a>		

Callback	1	2
<a href="#">CWDoVisualDifference</a>		X
<a href="#">CWCompletionRatio</a>		X
<a href="#">CWSetCommandDescription</a>		X

1: not supported in all older IDEs (Pro2 and earlier)

2: not supported in V1 compatibility mode.

These callbacks will be discussed below. The first argument to every callback routine is the `context` argument sent to the plug-in's main entry point when it is called by the IDE. If you call a callback and you are running in a mode where it is not allowed, its result will be [cwErrInvalidCallback](#).

### **CWAllowV1Compatibility**

```
CW_CALLBACK CWAllowV1Compatibility
(CWPluginContext context,
 Boolean canHandleV1Mode, Boolean *isV1);
```

This callback must be called as the first thing done in a Version 7 or later plug-in. It ensures functionality (either through compatibility emulation by the IDE, or by failing to load) in pre-Pro 4 IDEs.

The `canHandleV1Mode` boolean is an argument that is set to either true or false. If it is set to true, then it allows the plug-in to function in version 1 compatibility mode if an IDE calls it which does not recognize the Version 7 API or newer. If it is false and if the plug-in is called by an IDE that cannot run Version 7 of the API, the function will return an error which, when sent back to the IDE, will prevent the IDE from attempting to use the plug-in (any attempts would cause a system crash due to different calling conventions).

The `isV1` argument is a pointer to a Boolean value which will be set to true if the plug-in is called in V1 mode and false if called with Version 7 or greater. If it is true, the plug-in is running in V1 compatibility mode and certain callbacks will be invalid and specific requests not sent.

If this function returns an error, the plug-in should immediately return with the error value. If it does not return immediately, a

system crash may result if the plug-in is accidentally installed on a pre-Pro 4 IDE and it cannot run in V1 compatibility mode.

**CWGetComment**

```
CW_CALLBACK CWGetComment  
    (CWPluginContext context, const char *pPrompt,  
     const char *pComment, const long lBufferSize);
```

This function can be called by a plug-in to display a comment dialog. This is used for getting a line of comment text from the user.

The `pPrompt` argument is a pointer to a string that is used as the prompt above an area where the user enters their comment.

The `pComment` argument is a pointer to a buffer where the comment is to be stored. The buffer must be allocated before the function is called.

The `lBufferSize` argument indicates the maximum length of the buffer in bytes.

The function will return [cwNoErr](#) if the operation was successful. If the user cancelled the operation by hitting the Cancel button in the dialog they are presented with, then the function will return [cwErrUserCancelled](#).

**CWFfileStateChanged**

```
CW_CALLBACK CWFfileStateChanged  
    (CWPluginContext context, const CWFfileSpec  
     *file,  
     CWVCSCheckoutState eCheckoutState,  
     const CWVCSVersion version);
```

This is a deprecated version of `CWVCSStateChanged`. The two routines are identical except that `CWVCSStateChanged` passes the `version` parameter by reference rather than by value.

- 
- NOTE** This routine is no longer declared in the version 8 API headers, but is still included in the plug-in API library, for backward compatibility. The only plug-ins that should use this routine are old plug-ins. In order to use this routine, plug-ins must define `CWFfileStateChanged` as `extern`.
-

### CWVCSStateChanged

```
CW_CALLBACK CWVCSStateChanged
    (CWPluginContext context, const CWFfileSpec*
     file,
      CWVCSCheckoutState eCheckoutState,
      const CWVCSVersion* version);
```

This function must be called whenever the VCS plug-in changes a file on the disk. This includes if either its checkout state changes due to changes in the VCS database, or if the local file's information is changed (e.g. modification date/locked state) by the plug-in. Without making this call, the IDE will not know that the plug-in has changed the file in any way.

The `file` argument specifies the file whose state has changed.

The `eCheckoutState` argument specifies the new checkout state of the file. It is one of the following constants:

```
enum // checkout state
{
    cwCheckoutStateUnknown
        // unknown
    cwCheckoutStateNotCheckedOut
        // not checked out
    cwCheckoutStateCheckedOut
        // checked out
    cwCheckoutStateNotInDatabase
        // not in database
    cwCheckoutStateMultiplyCheckedOut
        // multiply checked out
    cwCheckoutStateNotCheckedOutShared
        //not checked out and shared
    cwCheckoutStateCheckedOutShared
        // checked out and shared
    cwCheckoutStateMultiplyCheckedOutShared
        // multiply checked out and shared
    cwCheckoutStateNotCheckedOutBranched
        // not checked out and branched
    cwCheckoutStateCheckedOutBranched
        //checked out and branched
    cwCheckoutStateMultiplyCheckedOutBranched
        // checked out and branched
```

```
    cwCheckoutStateNotCheckedOutSharedBranched
        // not checked out, shared and branched
    cwCheckoutStateCheckedOutSharedBranched
        // checked out, shared and branched
    cwCheckoutStateMultipleCheckedSharedOutBranched
        // checked out, shared and branched
    cwCheckoutStateCheckedOutExclusive
        // exclusively checked out
    cwCheckoutStateCheckedOutExclusiveShared
        // exclusively checked out and shared
    cwCheckoutStateCheckedOutExclusiveBranched
        // exclusively checked out and branched
    cwCheckoutStateCheckedOutExclusiveSharedBranch
        // exclusively checked out,
        // shared and branched
    cwCheckoutStateMultiplyCheckedOutMask
        // multiply checked out mask
    cwCheckoutStateSharedMask
        // shared mask
    cwCheckoutStateBranchedMask
        // branched mask
    cwCheckoutStateExclusiveMask
        // exclusive mask
};
```

The `version` argument contains the new version information for the file. The `version` is a `CWVCSVersion` structure:

```
typedef struct CWVCSVersion // version
{
    CWVCSVersionForm eVersionForm;
        // version form
    CWVCSVersionData sVersionData;
        // version data
}
```

The `version form` is one of the following constants:

```
enum // version form
{
    cwVersionFormNone      // no record
    cwVersionFormNumeric   // intergral numeric
    cwVersionFormAlpha     // alphabetic
    cwVersionFormDate      // date / time
```

```
        cwVersionFormLabel    // label
};
```

which describes which member of the `CWVCSVersionData` union is occupied:

```
typedef union CWVCSVersionData // version data
{
    unsigned long numeric;      // integral numeric
    char *pAlpha;              // alphabetic
    CWVCSDateTime date;        // date / time
    char *pLabel;               // label
}
```

These version forms correspond to the following types of version information:

#### **1. None**

No version information for this item is provided.

#### **2. Numeric**

In the numeric case the version is an integer, specifically an `unsigned long`.

#### **3. Alphanumeric**

In this case the version is represented by a character string. An example of this would be “`1.3.6a2`”.

#### **4. Date**

In the date case the standard library `struct tm` format is used. Any function from the standard library in `<time.h>` can be used to work with these times and do time differences.

#### **5. Label**

The label case is essentially the same as the alphanumeric case, but is included for clarity.

#### **CWDoVisualDifference**

```
CW_CALLBACK CWDoVisualDifference
    (CWPluginContext context,
```

```
const CWFfileSpec *file1, const char *pTitle1,
const char *pText1, unsigned long lengthText1,
const CWFfileSpec *file2, const char *pTitle2,
const char *pText2, unsigned long lengthText2);
```

This callback allows the VCS to ask the IDE to display a window containing the differences between the two passed files. If a file is on disk, its arguments (either the first or second group) should be:

```
fileX = pointer to CWFfileSpec for the file
pTitleX = NULL
pTextX = NULL
lengthTextX = 0
```

If a file is not on disk, for example, if its contents were retrieved from the server, its arguments would be:

```
fileX = NULL
pTitleX = pointer to name of file/component
pTextX = pointer to text of file
lengthTextX = length of text pointed to in bytes
```

Of the two files, the second file is the one that will be modified. Thus, if one of the files is local, you should always pass it as file2 so any differences of the local copy to the one stored in the database can be applied.

### **CWCompletionRatio**

```
CW_CALLBACK CWCompletionRatio
(CWPPluginContext context, int totalItems,
 int completedItems);
```

When plug-ins are processing a list of items or performing recursive operations and can determine both the total number of items that need to be processed as well as how many have been completed, the plug-in can use this routine to report this completion information back to the IDE. The IDE will then use this information to display a visual progress bar to the user indicating how far along the VCS operation is.

If your plug-in does not call this function, the IDE will display an indeterminate animated progress bar instead.

### **CWGetProjectFileSpecifier**

```
CW_CALLBACK CWGetProjectFileSpecifier  
(CWPluginContext context,  
 CWFileSpec *projectFileSpec);
```

This callback will fetch the [CWFileSpec](#) that is for the project file the plug-in is being called with requests for.

### **CWIsAdvancedRequest**

```
CW_CALLBACK CWIsAdvancedRequest  
(CWPluginContext context,  
 Boolean *isAdvanced);
```

The CodeWarrior IDE will send requests to the plug-in in four separate modes: regular, advanced, recursive, and command negotiation. Regular mode is indicated by all of these mode callbacks returning `false`.

This callback can be used to determine if the request was sent in the advanced mode. If the request was sent in advanced mode, the VCS plug-in should provide the user with a dialog box which contains the more advanced options for a command. Example: for a history command the advanced request could cause the plug-in to display a dialog filtering the history based by user.

### **CWIsRecursiveRequest**

```
CW_CALLBACK CWIsRecursiveRequest  
(CWPluginContext context,  
 Boolean *isRecursive);
```

This callback can be used to determine if the request was sent in the recursive mode. If the request was sent in recursive mode and a directory is encountered in the `CWVCSItem` list of items to process, the plug-in should apply the command to all files in the directory and recursively on any directories it contains.

### **CWIsCommandSupportedRequest**

```
CW_CALLBACK CWIsCommandSupportedRequest  
(CWPluginContext context,  
 Boolean *isCommandSupported);
```

The command supported mode is done during request negotiation. If the command is sent in this mode, the callback should set the command status to one of the following constants:

```
cwCommandStatusCommandUnknown  
    // plug-in does not recognize command  
cwCommandStatusUnsupported  
    // plug-in cannot perform command  
cwCommandStatusSupported  
    // plug-in can perform command
```

If a command is sent in the command supported mode, it is not actually carried out by the plug-in. It is only a query to see if the plug-in has the capability to perform a certain type of command at that time.

### **CWSetCommandDescription**

```
CW_CALLBACK CWSetCommandDescription  
(CWPluginContext context,  
 CWVCSCommandDescription *description);
```

Plug-ins can use this callback to return more information about the command to the IDE. These descriptions can be used by the IDE to better reflect the appropriate terminology for your command in the VCS menus and dialogs.

This callback has an effect only in the negotiation stage, and if you call it for a non-negotiation request it will have no effect. If you receive a negotiation request, at that time you can use this callback to return to the IDE the special strings used in the user interface.

You specify information about a command by using a `CWVCSCommandDescription` structure:

```
enum  
{  
    cwCommandDescriptionVersion = 1  
};  
  
typedef struct CWVCSCommandDescription  
{  
    long    version;  
    char    menuItem[ 40 ];  
    char    progressMessage[ 200 ];
```

```
} CWVCSCommandDescription;
```

When creating a command description structure, you must set its version field to an appropriate non-zero value. The current version of the command description structure will always be stored in the constant `cwCommandDescriptionVersion`.

In version 1 of the command description, you can set the `menuItem` string to be a short one or two word description of your command. The IDE can then use this short string in the VCS menus. The `progressMessage` string is used to display more detailed information about the command in the VCS progress dialog. If the plug-in does not call [CWShowStatus](#), the `progressMessage` string will be displayed instead of the plug-in's status reports.

### **CWGetDatabaseConnectionInfo**

```
CW_CALLBACK CWGetDatabaseConnectionInfo  
(CWPluginContext context,  
 CWVCSDatabaseConnection *dbConnection);
```

This callback allows the plug-in to obtain the current database connection information, including the path to the database directory (if applicable), path to the user's local root directory, username, and password. The resulting structure is of the form:

```
typedef struct CWVCSDatabaseConnection  
{  
    CWFileSpec    sDatabasePath;  
    CWFileSpec    sProjectRoot;  
    char          *pUsername;  
    char          *pPassword;  
};
```

### **CWGetCommandStatus**

```
CW_CALLBACK CWGetCommandStatus  
(CWPluginContext context,  
 CWVCSCommandStatus *status);
```

This callback returns what the IDE believes is the current status of the operation. It should be changed with [CWSetCommandStatus](#) by the plug-in to reflect the overall state of completion of the operation. It is one of these following constants:

```
enum // command status
{
    cwCommandStatusCommandUnknown,
    cwCommandStatusUnknown,
    cwCommandStatusUnsupported,
    cwCommandStatusSupported,
    cwCommandStatusSucceeded,
    cwCommandStatusFailed,
    cwCommandStatusPartial,
    cwCommandStatusCancelled,
    cwCommandStatusConnectionLost,
    cwCommandStatusInvalidLogin
};
```

**CWSetCommandStatus**

```
CW_CALLBACK CWSetCommandStatus
(CWPluginContext context,
 CWVCSCommandStatus status);
```

Use this callback to change the current overall completion status of the current command.

Completion states for individual items should be set using [CWSetVCSItem](#).

**CWGetVCSItemCount**

```
CW_CALLBACK CWGetVCSItemCount
(CWPluginContext context, unsigned long
 *count);
```

When the IDE calls a VCS plug-in, it can call certain commands with lists of items to be processed. These items can be files or directories to be operated on in either a regular, advanced, or recursive mode as determined from the mode callbacks listed above.

This callback is used to retrieve the number of items there are to be processed for a given request.

**CWGetVCSItem**

```
CW_CALLBACK CWGetVCSItem
(CWPluginContext context, long index,
 CWVCSItem *item);
```

This callback is used to retrieve the nth item to be processed, indexed from zero. The item is returned as a structure of the following form:

```
typedef struct CWVCSItem
{
    CWFfileSpec           fsItem;
    CWVCSItemStatus       eItemStatus;
    CWVCSVersion          version;
    // following field added for version 8 API:
    CWVCSCheckoutState   eCheckoutState;
}
```

The `fsItem` member of the structure contains the [CWFfileSpec](#) corresponding to the file or directory to be processed. The `eItemStatus` member contains the completion state of the operation on that member, which is one of the following constants:

```
enum // item status
{
    cwItemStatusUnprocessed,
    cwItemStatusUnknown,
    cwItemStatusSucceeded,
    cwItemStatusFailed,
    cwItemStatusCancelled
};
```

After the plug-in finishes processing an item, it should change `eItemStatus` to the appropriate code and call [CWSetVCSItem](#) with the updated `CWVCSItem` structure.

`version` is a `CWVCSVersion` structure in which the IDE returns the current version of the file. The version structure is described in the section for the [CWVCSStateChanged](#) callback.

`eCheckoutState` indicates the current check out state of the file, and is one of the constants listed under [CWVCSStateChanged](#).

### **CWSetVCSItem**

```
CW_CALLBACK CWSetVCSItem
(CWPPluginContext context, long index,
 CWVCSItem *item);
```

This callback is used by a plug-in to update information on an item that has been processed. The items are indexed from 0. See [CWGetVCSItem](#) for more information.

**CWGetVCSPointerStorage**

```
CW_CALLBACK CWGetVCSPointerStorage  
(CWPluginContext context, void **storage);
```

Versions 7 and later of the VCS API allow plug-ins to have one non-volatile storage area for a pointer that will be unique for each thread the plug-in is called on. This value can be changed by the plug-in and will be passed in for all consecutive calls to the plug-in that execute on the same thread until the plug-in changes it again. By examining its value, a plug-in can determine which thread it is being called from.

This callback allows you to retrieve the contents of this storage area.

**CWSetVCSPointerStorage**

```
CW_CALLBACK CWSetVCSPointerStorage  
(CWPluginContext context, void *storage);
```

This callback allows you to set the contents of the per-thread storage area described above.

## VCS Commands

There are two (2) modes of operation that are used by the IDE to communicate with the VCS. These are negotiation and execution.

### Negotiation Mode

After performing an initialization sequence, the IDE will query the VCS to determine which commands are supported. This information is used to construct and enable the VCS menu within the IDE. In this mode the command passed is not to be executed, merely responded to as to whether the VCS supports the command. Every command sequence will be queried. If the [CWIsCommandSupportedRequest](#) callback results in a true value, then this mode is being used. The [CWIsAdvancedRequest](#) callback will be set to true to indicate that information about advanced options is requested. At this point the plug-in would display a dialog to get this information from the user. The

[`CWIsRecursiveRequest`](#) will result in true to indicate that the operation is a recursive one. The items used for recursive operations will be directories.

It is important to note that there are only two (2) valid responses for the command supported query (`cwCommandStatusSupported` and `cwCommandStatusUnsupported`). During execution mode, if a request is made which was not negotiated, `cwCommandStatusCommandUnknown` should be returned.

### **Execution Mode**

In execution mode, the IDE asks the VCS plug-in to perform version control services.

### **Command Types**

There are four (4) types of execution mode commands which may be passed to the VCS:

- [Initialization / Termination](#)
- [Database Connection / Disconnection](#)
- [Queries](#)
- [Information](#)
- [File Processing](#)

### **Advanced Options**

When the user specifies advanced options, the plug-in presents them with a dialog, gets the options and then handles the request, processing each item with the specified options applied.

### **Reporting Status**

Command status is reported two (2) ways. The first is in the command's return status which set with the [`CWSetCommandStatus`](#) callback. As a rule of thumb, the value of this should start out as `cwComandStatusUnknown`. If all items are processed successfully, `cwCommandStatusSucceeded` should be returned. If none of the items are processed successfully, `cwCommandStatusFailed` should be returned. Finally, if some, but not all, of the items are processed successfully, `cwCommandStatusPartial` should be returned. If the user

cancels the operation, `cwCommandStatusCancelled` should be returned.

The second method provides additional detail for the first. This information is in the `eItemStatus` field of a `CWVCSItem` and is returned to the IDE by passing it in the argument to [CWSetVCSItem](#) after processing is finished. This field should be initialized for all items to `cwItemStatusUnknown`. Prior to processing all should be set to `cwItemStatusProcessed`. As processing progresses each item should be set to `cwItemStatusSucceeded`, `cwItemStatusFailed` or `cwItemStatusCancelled`. In this way, the IDE will be able to determine just how far along things got if things go bad.

## VCS Requests

The following is a listing of all of the requests sent to a VCS plug-in followed by specific descriptions:

```
enum // IDE requests to VCS plug-in
{
    reqInit,
    reqTerminate,
    reqPrefsChange,           // *
    reqIdle,                  // *
    reqDatabaseConnect,
    reqDatabaseDisconnect,
    reqDatabaseVariables,
    reqFileAdd,
    reqFileCheckin,
    reqFileCheckout,
    reqFileComment,
    reqFileDelete,
    reqFileDestroy,
    reqFileDifference,
    reqFileGet,
    reqFileHistory,
    reqFileLabel,
    reqFileProperties,
    reqFilePurge,
    reqFileRename,
    reqFileRollback,
```

```
    reqFileStatus,  
    reqFileUndoCheckout,  
    reqFileVersion,  
    reqFileBranch,           // *  
    reqFileShare,            // *  
    reqFileView              // *
```

} ;

---

**NOTE** Starred (\*) requests are not sent in V1 compatibility mode.

---

## Initialization / Termination

There are a pair of commands used to inform the VCS plug-in it is being loaded or unloaded.

---

**WARNING!**

Only the standard memory callbacks, [CWGetAPIVersion](#), [CWGetPluginRequest](#), [CWDonePluginRequest](#), and [CWGetIDEInfo](#) are valid during initialize and termination requests.

---

### reqInitialize

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command informs the VCS plug-in that it has just been loaded.

Header CWPlugins.h

Returns status using  
CWSetCommandStatus();  
CWDonePluginRequest();  
entry point return value

See Also [“reqInitialize” on page 192](#)

[“reqInitialize Request” on page 78](#)

### reqTerminate

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command informs the VCS plug-in that it is about to be unloaded.

Header CWPlugins.h

## Version Control System Plug-in API

### Database Connection / Disconnection

---

Returns status using  
CWSetCommandStatus( );  
CWDonePluginRequest( );  
entry point return value

See Also  
[“reqTerminate” on page 193](#)  
[“reqTerminate Request” on page 79](#)

## Database Connection / Disconnection

There are three commands for dealing with database connections:

### reqDatabaseConnect

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to connect to its database.  
  
The connection information is specified in `pDatabaseConnection`. This information should be stored along with any other session specific information in the `context` field. As with all IDE plug-ins, this will be retained for the duration of the connection and passed on all subsequent calls. It will not be modified by the IDE however. This memory must be allocated by the plug-in.

Header `DropInVCS.h`

Callbacks used `CWGetDatabaseConnectionInfo()`;

Returns status using `CWSetCommandStatus()`

### reqDatabaseDisconnect

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to disconnect from its database.

At this time the plug-in should release the memory being used to store the session information in `context`.

Header `DropInVCS.h`

Returns status using `CWSetCommandStatus()`;

## Queries

There are two commands for dealing with queries :

### **reqAbout**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description      This command instructs the VCS plug-in to output its identification information.

It is not required that the plug-in display a dialog with this information. Creating an IDE text view is fine and has the added advantage of being printable.

Header      CWPlugins.h

Callbacks available      **memory routines**  
**CWPreDialog()**  
**CWPostDialog();**

Returns status using      CWDonePluginRequest();

### **reqDatabaseVariables**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description      This command instructs the VCS plug-in to output its initialization variables.

Header      DropInVCS.h

Returns status using      CWSetCommandStatus();

## Information

There are three commands for dealing with information updates:

### **reqIdle**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description      This command informs the VCS plug-in that time is available to it to be used to perform periodic operations.

Header      CWPlugins.h

Callbacks available      **memory callbacks**

## Version Control System Plug-in API

### File Processing

---

```
CWPreDialog();
CWPostDialog();
CWGetAPIVersion();
CWGetIDEInfo();
```

Returns status using `CWDonePluginRequest();`

#### **reqPrefsChange**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command informs the VCS plug-in that one of its preference panels has changed.

Header `CWPlugins.h`

Callbacks available All universal plug-in callbacks.

Returns status using `CWDonePluginRequest();`

## File Processing

The following commands instruct the VCS plug-in to manipulate a list of file in the database.

### Item Data and Status

Commands which may change the checkout state of files must update (if possible) the version field of pItemData in pItemList. When passed in the version type none and the version data zero.

The status of items will be unprocessed on entry and must be set on exit.

The overall command status should only be set to success or failure if all items have the corresponding status. When the user cancel status is used the IDE will walk the items to determine the appropriate action to take for each.

#### **reqFileAdd**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to add a list of file to the database.

Header    DropInVCS.h

Returns status using  
 CWSetCommandStatus( );  
 CWSetVCSItem( );

### **reqFileBranch**

<b>REQUIRED</b>	<b>ADVANCED</b>	<b>RECURSIVE</b>
-----------------	-----------------	------------------

Description    This command instructs the VCS plug-in to branch in a list of files in the database.

Header    DropInVCS.h

Returns status using  
 CWSetCommandStatus( );  
 CWSetVCSItem( );

### **reqFileCheckin**

<b>REQUIRED</b>	<b>ADVANCED</b>	<b>RECURSIVE</b>
-----------------	-----------------	------------------

Description    This command instructs the VCS plug-in to check in a list of files into the database.

Header    DropInVCS.h

Returns status using  
 CWSetCommandStatus( );  
 CWSetVCSItem( );

### **reqFileCheckout**

<b>REQUIRED</b>	<b>ADVANCED</b>	<b>RECURSIVE</b>
-----------------	-----------------	------------------

Description    This command instructs the VCS plug-in to check out a list of files from the database.

Header    DropInVCS.h

Returns status using  
 CWSetCommandStatus( );  
 CWSetVCSItem( );

### **reqFileComment**

<b>REQUIRED</b>	<b>ADVANCED</b>	<b>RECURSIVE</b>
-----------------	-----------------	------------------

Description    This command instructs the VCS plug-in to change the comment of a list of files in the database.

Header    DropInVCS.h

## Version Control System Plug-in API

### File Processing

---

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

#### reqFileDelete

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to delete of a list of files from the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

#### reqFileDestroy

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to destroy of a list of files from the database. This is the same as a delete followed by a purge.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

#### reqFileDifference

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to output a difference listing between a list of files in the database and those local.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

#### reqFileGet

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to get a list of files from the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );

CWSetVCSItem( );

#### **reqFileHistory**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to output a history listing for a list of files in the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

#### **reqFileLabel**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to label a list of files in the database. The start version field is used to indicate label to be applied.

Header DropInVCS.h

Returns status using CWSetCommandStatus( );

#### **reqFileProperties**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to output the properties of a list of files in the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

#### **reqFilePurge**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to permanently remove a list of files from the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus( );  
CWSetVCSItem( );

## Version Control System Plug-in API

### File Processing

---

#### reqFileRename

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to rename a list of files in the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus();  
CWSetVCSItem();

#### reqFileRollback

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to rollback a list of files in the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus();  
CWSetVCSItem();

#### reqFileShare

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to share in a list of files in the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus();  
CWSetVCSItem();

#### reqFileStatus

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to output the status of a list of files in the database.

Header DropInVCS.h

Returns status using  
CWSetCommandStatus();  
CWSetVCSItem();

### **reqFileUndoCheckout**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to cancel the checkout of a list of files in the database.

Header DropInVCS.h

Returns status using CWSetCommandStatus( );  
CWSetVCSItem( );

### **reqFileVersion**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to retrieve the version and checkout state of a list of files in the database.

Header DropInVCS.h

Returns status using CWSetCommandStatus( );  
CWSetVCSItem( );

### **reqFileView**

REQUIRED	ADVANCED	RECURSIVE
----------	----------	-----------

Description This command instructs the VCS plug-in to display the contents of a list of files from the database.

Header DropInVCS.h

Returns status using CWSetCommandStatus( );  
CWSetVCSItem( );

## **VCS API Version 1 Compatibility**

The plug-in libraries that you link against when writing Version 7 and later VCS plug-ins do allow for backwards compatibility. That is, running a V7 or later plug-in under an IDE that was released prior to the Pro 4 IDE, from CodeWarrior 10 and up. This compatibility comes at a price, however, in that you must only use specific callbacks and can't expect to receive certain requests.

The following sections detail the overall behavioral differences, which callbacks are supported when running in V1 compatibility mode, and which requests are not sent in V1 compatibility mode.

## Version 1 VCS API

If you are writing a V7 or later plug-in and want it to run in V1 compatibility mode, you must pass `true` as the argument to all of the calls to [`CWAllowV1Compatibility`](#) and assume the behavior of a V1 plug-in, not utilize any of the unmapped callbacks, and not expect to receive any requests specific to later versions of the API.

### Installing on a Pre-Pro 4 CodeWarrior Installation

To run a V7 or later plug-in properly on a pre-Pro 4 CodeWarrior IDE, the DLL “`PluginLib4.dll`” on the user’s system must be either replaced with or created with the one contained in the SDK to gain access to the new VCS callbacks and compatibility code. Without it, V7 and later VCS plug-ins will not function.

### Behavioral Differences

When a VCS plug-in is running in V1 compatibility mode, it should expect the following major behavioral differences:

- no idle time will be yielded to it
- no notification will be sent of a preferences change
- all files will be prepared before calls to the main entry point (no Pre/Post file actions necessary)
- no events will be eaten

When writing a plug-in that is intended to be run in V1 compatibility mode, there are two choices. You can implement work-arounds for the lack of these functions yourself, such as installing a Time Manager task to mimic an idle event, and use them selectively by testing the `isV1` return value of [`CWAllowV1Compatibility`](#).

The second option is to write your plug-in not assuming any behavior or callbacks that is not supported in V1 compatibility mode. In order to set the behavior, you would specify the following

to be your capability flags in the Flag resource or entry point return structure:

```
flags.dropinflags =  
    vcsDoesntPrePostSingleFile |  
    vcsDoesntPrePostRecursiveFile |  
    vcsRequiresEvents;
```

Note: all the constants, version structures (struct tm), VCS items, and the file structures (FSSpec on Mac, full path on Windows) will be in version 7 (or later) format. Take special care when porting an older plug-in that you are aware of the differences in the structures and change how you access their members accordingly.

## Supported Callbacks

The callbacks that you can use when running in V1 compatibility mode are limited by the functionality of the V1 VCS API. Only the following callbacks may be used:

Standard callbacks:

- [CWGetPluginRequest](#)
- [CWDonePluginRequest](#)
- [CWGetAPIVersion](#)
- [CWReportMessage](#)
- [CWAlert](#)
- [CWShowStatus](#)
- [CWUserBreak](#)
- [CWGetNamedPreferences](#)
- [CWCreateNewTextDocument](#)
- [CWAllocMemHandle](#)
- [CWFreeMemHandle](#)
- [CWGetMemHandleSize](#)
- [CWResizeMemHandle](#)
- [CWLockMemHandle](#)
- [CWUnlockMemHandle](#)
- [CWPreDialog](#)

- [CWPostDialog](#)

VCS-specific callbacks:

- [CWAllowV1Compatibility](#)
- [CWGetComment](#)
- CWUpdateCheckoutState
- [CWDоВизуальнаяРазница](#) (Pro 2 and later only)
- [CWGetProjectFileSpecifier](#)
- [CWIsAdvancedRequest](#)
- [CWIsRecursiveRequest](#)
- [CWIsCommandSupportedRequest](#)
- CWGetOverridingMenuString
- CWSetOverridingMenuString
- [CWGetDatabaseConnectionInfo](#)
- [CWGetCommandStatus](#)
- [CWSetCommandStatus](#)
- [CWGetVCSItemCount](#)
- [CWGetVCSItem](#)
- [CWSetVCSItem](#)

No callbacks aside from [CWGetPluginRequest](#), [CWDonePluginRequest](#), [CWPreDialog](#), [CWPostDialog](#), and [CWGetAPIVersion](#) can be used during initialization and termination.

If you try to use a callback that is not supported in V1 compatibility mode, it will return an error and not have any effects.

## Unsupported Requests

The following requests will **not** be received by your plug-in when it is running in V1 compatibility mode:

- [reqIdle](#)
- [reqPrefsChange](#)
- [reqFileBranch](#)
- [reqFileShare](#)

- [reqFileView](#)

## Porting a Version 1 VCS Plug-in to Version 7 or Later API

In order to ease the process of porting an existing Version 1 VCS plug-in to versions 7 and later of the VCS API, a header file `DropInVCS_V1Compatibility.h` can be included. This header file will allow you to avoid having to rename all of the constants and structure types that are supported in the old Version 1 API. You will need to change your logical structure for the following structures whose formats have changed:

- `CWVCSDateTime`
- `CWVCSItemData`
- `CWVCSFileSpecifier`

In general, there are two options for porting a version 1 plug-in to the new API. One option which may be easiest for large plug-ins is to create your own version 1 structures from the old header and translate from the new accessor to the old parameter block structure on entry and reverse on exit. You will also need to implement your own glue routines for the callbacks that accept version 1 structures and translate them to version 7 or later structures when calling back to the IDE. Although this will allow existing code to work, care must be taken when using newer callbacks directly in the code, as your plug-in will be modifying your translated parameter block, not the actual information that is used to communicate with the IDE.

The second option is to get rid of all of the version 1 callbacks and parameter block references and move your code over to use the version 2 API directly. The steps in general for this process are:

- 1) Change the callbacks you use to the new VCS callbacks and standard plug-in callbacks.
- 2) Eliminate any access of a parameter block and replace with the accessor routines.
- 3) Insert calls to [CWPreFileAction](#) and [CWPostFileAction](#) callbacks if desired to allow for future compatibility.

- 4) Modify the format of your main entry point to make the required calls to [CWAllowV1Compatibility](#) and [CWDonePluginRequest](#).
- 5) Modify your Flag resource or entry point to use the new Version 3 Flag format set to the value specified in the prior section.
- 6) Look at each point where you use a CWVCSDateTime and CWVCSItem structure to be sure that you are using the proper version 7 (or later) structures.

You can look at the tables below for mappings of old callbacks to new callbacks and parameter block entries to accessors.

### V1 to V7 Callback Mappings

The following are the callbacks to be used in V7 for the function pointer entries in the V1 parameter block:

Version 1 Callback	Version 7 Callback
pVCSPreDialogRoutine	CWPreDialog
pVCSPostDialogRoutine	CWPostDialog
pVCSMessageOutputRoutine	CWReportMessage
pVCSReportProgressRoutine	CWShowStatus
pVCSGetPreferencesRoutine	CWGetNamedPreferences
pVCSCreateDocumentRoutine	CWCreateNewTextDocument
pVCSYieldTimeRoutine	CWUserBreak
pVCSGetCommentRoutine	CWGetComment
pVCSUpdateCheckoutStateRoutine	CWFfileStateChanged

### V1 Param Block Members to Accessor Mappings

V1 PB Member	Retrieval Accessor	Assignment Accessor
request	CWGetPluginRequest	X X X X X X X X X X X X X X X X
version	CWGetAPIVersion	X X X X X X X X X X X X X X X X
context	CWGetVCSPointerStorage	CWSetVCSPointerStorage
storage	X X X X X X X X X X X X X X X X	X X X X X X X X X X X X X X X X
targetfile	CWGetProjectFileSpecifier	X X X X X X X X X X X X X X X X

**Version Control System Plug-in API**  
*Porting a Version 1 VCS Plug-in to Version 7 or Later API*

---

bAdvanced	CWIIsAdvancedRequest	
bRecursive	CWIIsRecursiveRequest	
bCheckIfSupported	CWIIsCommandSupportedRequest	
eCommandStatus	CWGetCommandStatus	CWSetCommandStatus
sDatabasePath	CWGetDatabaseConnectionInfo	
sProjectRoot	CWGetDatabaseConnection-Info	
rcUsername	CWGetDatabaseConnection-Info	
rcPassword	CWGetDatabaseConnection-Info	
lItemCount	CWGetVCSItemCount	
pItemList	CWGetVCSItem	CWSetVCSItem

**Version Control System Plug-in API**  
*Porting a Version 1 VCS Plug-in to Version 7 or Later API*

---

# Browser Reference

---

This section documents the format of the browser data used by the CodeWarrior IDE class browser.

## Browser Reference Overview

This section documents the data structures and predefined symbols used in the browser data for the CodeWarrior IDE class browser.

- [Browser Records](#)—describes the format of a browser record, the unit used to describe a browsable item in source code.
- [Data Structures for the Browser](#)—describes the data structures used by the API to format data for the class browser.
- [Constants for the Browser](#)—describes the constant values used by the API to support the class browser.

For information on how to store browser data, see [“Storing Object, Resource, and Class Browser Data” on page 227](#).

---

**TIP** To examine a plug-in’s browser data output for debugging purposes, use the **Dump internal browse information after compile** checkbox in the **Build Extras** settings panel.

---

## Browser Records

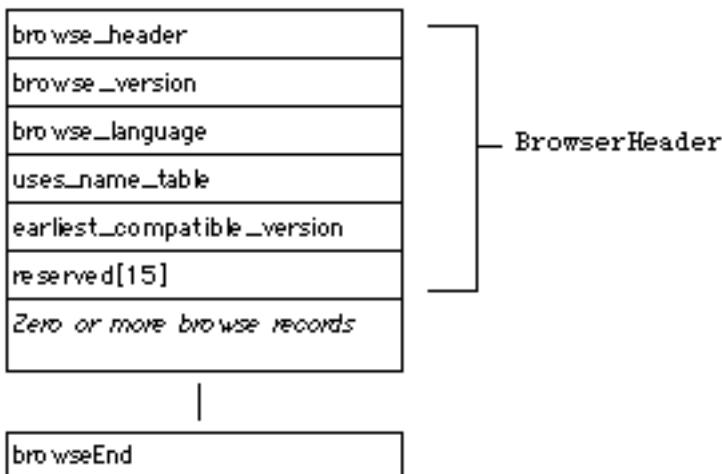
Compilers store browser data as a sequence of variable size records. This data is stored in the `browsedata` field of the [`CWObjectData`](#) structure passed to the IDE when calling [`CWStoreObjectData`](#). Browser data describes the symbols found by a compiler during compilation of source text. The IDE displays this information in the **Browser Contents**, **Class Hierarchy**, and **Class Browser** windows, accessible under the **Windows** menu.

- NOTE** Browser capabilities are only enabled when **Activate Browser** is checked in **Target Settings > Build Extras**.
- 

## Browse Header

The browser data stream starts with a [BrowseHeader](#), which describes the browser data to the IDE.

**Figure 13.1** **Browser data organization**



The header fields have the following meanings:

**browse\_header** 4 bytes—Contains the arbitrary value [BROWSE\\_HEADER](#), which indicates to the IDE that the browser data stream is valid.

**browse\_version** 4 bytes—Indicates the version of the browser data stream. This number changes as the browser data stream format is revised. plug-ins should normally specify the latest stream format, using [BROWSE\\_VERSION](#).

**browse\_language** 2 bytes—Specifies the language of the source file from which the browser symbol information was derived. See [ELanguage](#) for valid values.

**uses\_name\_table** 2 bytes—Reserved for Metrowerks internal use. plug-ins should always set this field to zero.

**earliest\_compatible\_version** 4 bytes—This field specifies the earliest browser stream format that this browser data is compatible

with. Plug-ins should normally store the value [BROWSE\\_EARLIEST\\_COMPATIBLE\\_VERSION](#) in this field.

---

**NOTE** The value of this field will typically match the value specified in `browse_version`, since browser data formats are rarely backward compatible.

---

**reserved** 60 bytes—15 four-byte integers reserved for future use. Should be set to zero.

The browser data stream ends with a single byte containing the value `browseEnd`. `browseEnd` is defined in the [EBrowserItem](#) enumeration.

## Browser Data Stream

As illustrated in [Figure 13.1](#), the header is followed by one variable-length record for each browsable item. Symbol records need not be presented in any particular order.

Browser records for individual symbols have variant definition, depending upon the symbol type. All symbol records start with common fields, with type-specific fields following.

---

**NOTE** Browser data is composed of records of variable size, without padding. Thus, the alignment of browser information varies, and need not be aligned on host CPU word boundaries. The easiest way to write browser data streams is with C++-style stream I/O.

---

The rest of this section discusses:

- [Fields For All Records](#)
- [Additional Fields For Functions](#)
- [Additional Fields For Classes](#)
- [Additional Field For Templates](#)
- [Additional Field For Global Variables](#)

## Fields For All Records

The common fields for all records describe the symbol's type, its point of declaration, and its name.

**Figure 13.2 A browse record**

type
contribFile
srcFile
startOffset
endOffset
reserved
simple name length
simple name
qualified name length
qualified name

*Additional fields if type is  
browseFunction, browse-  
Class, browseTemplate, or  
browseGlobal*

**type** 1 byte—Every record begins with the *type* field. This field specifies the type of item being described. The valid types are described in [“EBrowserItem” on page 589](#).

This field can also be set to a custom type, if a plug-in generates symbol records for nonstandard types of symbols. Custom type values should be assigned consecutively beginning with `browseCompSymbolStart`. See [“Compiler Browser Symbol Entry Point” on page 209](#) for more information.

**contribFile** 2 bytes—The file number of the file that should be considered the contributor of the item. For all record types except templates, this should be the same file as specified in *srcFile*. This must be a valid file number as returned by [CWFindAndLoadFile](#).

*contribFile* specifies the file contributing a fully-specified declaration of this syntactic entity. In the case of templates, this is

the file in which the template was instantiated, rather than the header in which the generic template definition appears.

The IDE uses this field when updating its internal browser information database. If contribFile is recompiled, the IDE will look for any items originally contributed by contribFile which don't appear in the newly generated browser data stream. These items will be removed from the IDE's database.

**srcFile** 2 bytes—The file number of the file containing the item's declaration. This must be a valid file number returned by [CWFindAndLoadFile](#).

**startOffset** 4 bytes—The zero-based offset of the start of the item's location in [srcFile](#). Use -1 to indicate an unknown or not-applicable offset.

**endOffset** 4 bytes—The zero-based offset of the start of the item's location in [srcFile](#). Use -1 to indicate an unknown or not-applicable offset.

**reserved** 4 bytes—Reserved for future use, must be zero.

**simple name length** 2 bytes—Length of following simple item name, excluding the terminating null character.

**simple name** variable length—The simple, unqualified name of the item as a variable-length, null-terminated C string. The length of this string is specified in [simple name length](#). The simple name of an item is the identifier by which it is normally specified in its parent language. It is the name a programmer would type to refer to the item.

For example, consider the following declaration:

**Listing 13.1 Simple name versus qualified name example**

---

```
class CExampleClass
{
public:
    CExampleClass ();
    virtual ~CExampleClass ();

    int Foo (int A);
```

```
int Foo (int A, int B);  
};
```

---

The simple, unqualified name of both methods named `Foo` is “`Foo`”. As this example shows, simple names need not be unique.

**qualified name length** 2 bytes—Length of following qualified item name, excluding the terminating null character. May be zero.

**qualified name** variable length—The qualified name of the item as a variable-length, null-terminated C string. The length of this string is specified in [qualified name length](#). This string is not stored if the [qualified name length](#) is zero.

The qualified name of an item specifies it uniquely, usually by including scope information, and any necessary distinguishing information within its scope. In listing 13.1, the first of the two overloaded `Foo` methods might have a qualified name of “`CExampleClass::Foo (int A)`”. The second `Foo` method might have a qualified name of “`CExampleClass::Foo (int A, int B)`”. The qualified names could also be “mangled” names.

The IDE places no restrictions on the content of a qualified name, but it should uniquely identify its associated symbol. The IDE uses the qualified name when searching for symbols in its internal browser symbol database, such as when adding and deleting symbols from the database. Qualified names for the same symbol should be consistent across compiles.

If `type` is `browseMacro`, `browseEnum`, `browseConstant`, `browseTypedef`, and `browsePackage`, or a compiler-specific browser symbol, then no additional fields are required for the browser record.

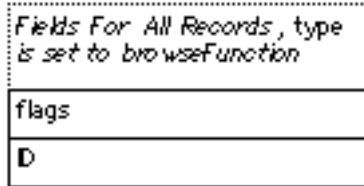
If `type` is `browseFunction`, `browseClass`, `browseTemplate`, or `browseGlobal`, then the record requires additional data, described here.

- [“Additional Fields For Functions” on page 579](#)—Additional fields for the `browseFunction` type of browser record.
- [“Additional Fields For Classes” on page 580](#)—Additional fields for the `browseClass` type of browser record.

- [“Additional Field For Templates” on page 584](#)—Additional fields for the browseTemplate type of browser record.
- [“Additional Field For Global Variables” on page 584](#)—Additional fields for the browseGlobal type of browser record.

## Additional Fields For Functions

**Figure 13.3 A browse record for a function**



For a list of fields required for all browser records, see [“Fields For All Records” on page 576](#).

The additional data for browseFunction records is:

**flags** 4 bytes—Specifies properties of the function. See listing 13.2 for the available flags. Flags may be combined.

**Listing 13.2 Function flags listing**

---

```
enum
{
    kStatic = 2,                      /* Static routine */
    kMember = 8,                      /* Class member function */
    kInline = 0x80,                   /* In line routine */
    kPascal = 0x100,                  /* Pascal routine */
    kAsm   = 0x200,                   /* Assembly routine */
};
```

---

**ID** 4 bytes—The member function ID for this function, so it can be matched with the class’ member function information. Store zero if this is not a member function.

Member function IDs uniquely identify the member functions of classes, and appear in the [Additional Fields For Classes](#). Member function IDs should be assigned starting with 1 and incremented for

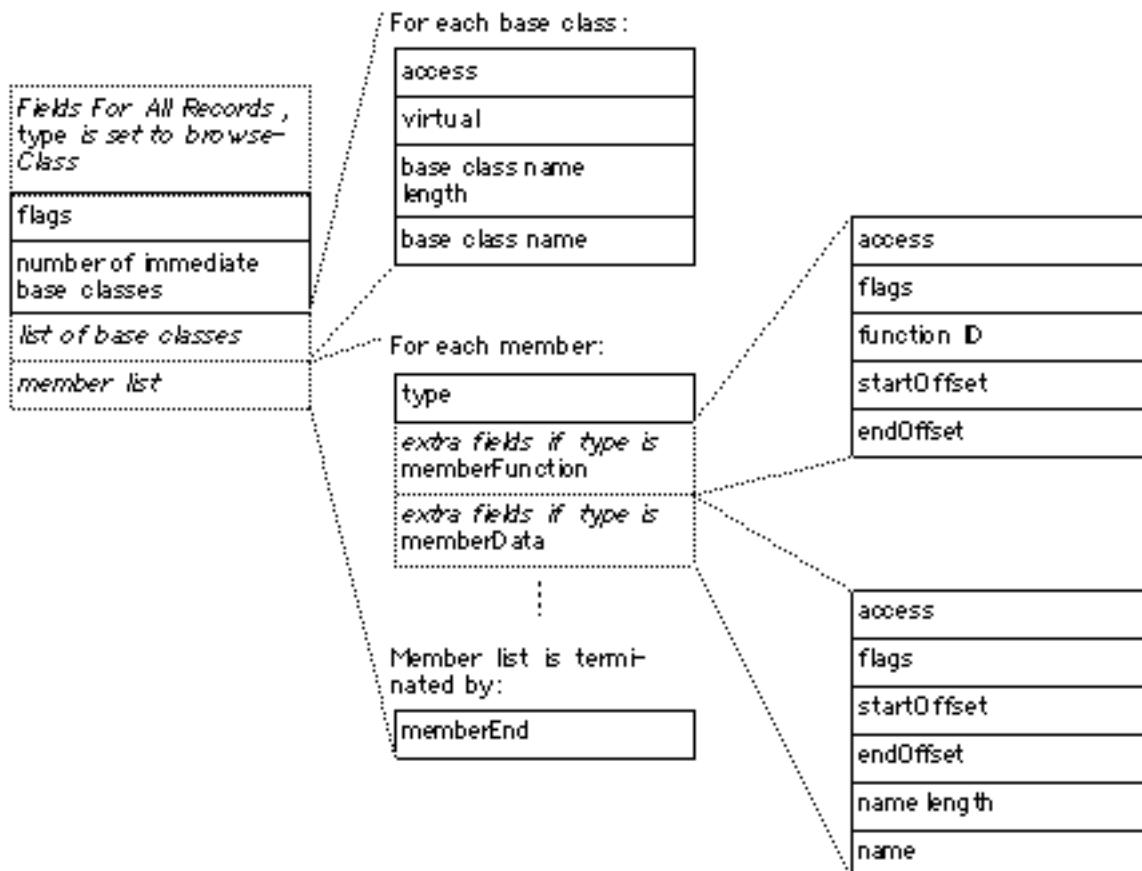
each subsequent member function. However, the order does not matter, and the numbers may change across compiles.

## Additional Fields For Classes

Browser records for classes include additional information, listing superclasses, data members, member functions, member access, and the source locations of member declarations.

As illustrated in [Figure 13.4](#), the browse data for a class starts with the usual information common to all browser records. This is followed by fixed-size class information, a variable-length base class list, and a variable-length member list.

**Figure 13.4 A browse record for a class**



**NOTE** The base class list length is determined from the count provided in the class information, whereas the length of the member list is determined by the presence of the `memberEnd` sentinel value.

---

For a list of fields required for all browser records, see “[Fields For All Records](#)” on page 576.

The additional data for `browseClass` records is:

**flags** 4 bytes—Specifies properties of the class as a whole. See listing 13.3 for valid flag values. Flags may be combined.

### **Listing 13.3 Class flags listing**

---

```
enum
{
    kAbstract      = 1,                      /* Abstract class */
    kFinal         = 4,                      /* Final class */
    kInterface     = 0x80,                   /* Java interface */
    kPublic        = 0x100,                  /* Public Java class */
};
```

---

**number of immediate base classes** 1 byte—Specifies the number of classes this class inherits from.

The count of base classes is immediately followed by one record for each class. Base class information should be omitted if there are no base classes. Base class records the following fields:

#### **For Each Base Class**

**access** 1 byte—Specifies the scoping relationship between this base class and the class inheriting from it. For valid values, see “[EAccess](#)” on page 588.

**virtual** 1 byte—Specifies whether this base class is virtual. True if this is a virtual base class.

**base class name length** 2 bytes—Specifies the length of the following base class name, excluding the terminating null character.

**base class name** variable length—Specifies the mangled or qualified name of this base class, terminated with a null character.

## Class Member List

For each member of the class, one class member record should appear following the base class records (if any). The class member record varies according to the member type (method or data member).

**member list** variable-length list of class members—Neither inherited data members nor methods should not be counted or described, only the ones declared by this class.

Each member record contains the following:

### For Each Class Member

**type** 1 byte—The type of class member. For valid values, see [“EMember” on page 591](#). When type is memberFunction or memberData, additional data follows. A type value of memberEnd signals the end of the member list for the current class.

The additional data for memberFunction is:

**access** 1 byte—Specifies the scope of this member function. For valid values, see [“EAccess” on page 588](#).

**flags** 4 bytes—Specifies properties of this member function. See listing [13.4](#) for valid flag values. Flags may be combined.

### **Listing 13.4 Member function flags listing**

---

```
enum
{
    kAbstract= 1,           /* Abstract/pure virtual */
    kStatic= 2,             /* Static member or function */
    kFinal= 4,              /* Final Java class/method/member */
    kVirtual= 0x400,         /* Virtual member function */
    kCtor= 0x800,            /* Is constructor */
    kDtor= 0x1000,           /* Is destructor */
    kNative= 0x2000,          /* Native Java method */
    kSynch= 0x4000,           /* Synchronized Java method */
};
```

---

**function ID** 4 bytes—The ID of this member function. The IDE uses member function IDs to match function records found

elsewhere in the browser data stream with method entries in class records. Method entries in class information minimally describe a method, and “point” to additional information in a corresponding function record using the function ID.

Function IDs must be unique and non-zero. Function IDs should be assigned to method functions – and their corresponding class method entries – starting with 1, incrementing sequentially. Function IDs can vary from compile to compile. Functions referred to by ID in a class member function entry need not appear in the browser stream before the class record.

**startOffset** 4 bytes—Zero-based offset from the start of the class declaration file (the `srcFile` specified for the class). The offset specifies the location of the member function’s declaration within its class declaration.

**endOffset** 4 bytes—Zero-based offset from the start of the class declaration file (`srcFile`) of the end of the method’s declaration. If the member function is defined inline in the class declaration, then the range includes the definition.

The additional data for `memberData` is:

**access** 1 byte—Specifies the scope of this data member. For valid values, see [“EAccess” on page 588](#).

**flags** 4 bytes—Specifies properties of this data member. See listing [13.5](#) for valid flag values. Flags may be combined

### **Listing 13.5 Class flags listing**

---

```
enum

{
    kStatic = 2,           /* static member */
    kFinal= 4,            /* final Java class/method/member */
    kTransient = 0x80,     /* transient Java member */
    kVolatile = 0x100,     /* volatile Java or C++ member */
};
```

---

**startOffset** 4 bytes—Zero-based offset from start of the class declaration file (`srcFile`) to the beginning of the data member declaration.

## Browser Reference

### Additional Field For Templates

---

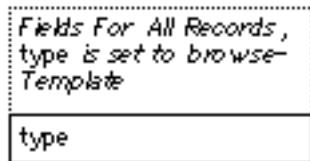
**endOffset** 4 bytes—Zero-based offset from start of the class declaration file (`srcFile`) to the end of the data member declaration.

**name length** 2 bytes—Length of following name, not including the terminating null character.

**name** Variable length—Name of data member, terminated with a null character.

## Additional Field For Templates

**Figure 13.5** A browse record for templates



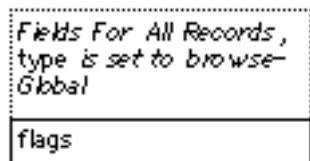
For a list of fields required for all browser records, see ["Fields For All Records" on page 576](#).

The additional data for records of type `browseTemplate` consists of a single byte:

**type** 1 byte. Specifies the type of the template (class or function). For valid values, see ["ETemplateType" on page 592](#).

## Additional Field For Global Variables

**Figure 13.6** A browse record for a global variable



For a list of fields required for all browser records, see ["Fields For All Records" on page 576](#).

The additional data for records of type `browseGlobal` is:

**flags** 4 bytes—A set of bit flags that specifies additional information about the global variable. See listing [13.6](#) for valid flag values.

#### **Listing 13.6 Global variable types listing**

---

```
enum
{
    kStatic = 2,                                /* Static global */
};
```

---

## **Data Structures for the Browser**

The plug-in API header '`MWBrowse.h`' defines the following data structures, used when generating browser data:

- [BrowseHeader](#)

---

**NOTE** `MWBrowse.h` includes definitions for an older browser data stream format. This format is no longer used or supported by the IDE. Only current data structures are described in this section.

---

### **BrowseHeader**

**Description** Describes a browser data stream.

**Prototype**

```
#include <MWBrowse.h>
typedef struct BrowseHeader {
    long          browse_header;
    long          browse_version;
    short         browse_language;
    short         uses_name_table;
    long          earliest_compatible_version;
    long          reserved[15];
} BrowseHeader;
```

**Fields** The fields in `BrowseHeader` are:

## Browser Reference

### *BrowseHeader*

---

<code>browse_header</code>	Contains the value of <a href="#">BROWSE HEADER</a> . This field is simply a distinctive value used by the IDE to ensure that the browser data stream is valid.
<code>browse_version</code>	Indicates the format of the browser data stream. Currently, compilers should store <a href="#">BROWSE VERSION</a> in this field.
<code>browse_language</code>	Specifies the language of the source file for which the browser data was generated. Valid language values are listed in <a href="#">“FLanguage” on page 590</a> .
<code>uses_name_table</code>	If the compiler stores true in this field, the compiler uses the CodeWarrior object code name table. Non-Metrowerks compilers should set this field to false.
<code>earliest_compatible_version</code>	Specifies the earliest version of the browser data stream format that this data is compatible with. plug-ins should use <a href="#">BROWSE EARLIEST COMPATIBLE VERSION</a> for this field, which will usually be the same as <a href="#">BROWSE VERSION</a> .
<code>reserved[15]</code>	Reserved for future use. Should be set to zero.

**Remarks** All browser data streams must start with a single `BrowseHeader`. The header describes properties of the stream to the IDE.

A valid browser data stream starts with a single `BrowseHeader`, which is followed by additional variable-length data records. Each record describes a single browsable item (class, function, global variable, or other browser-supported source element).

**See Also** [“Browser Records” on page 573](#)

[“BrowseHeader” on page 585](#)

## Constants for the Browser

This section documents the constants and enumerations defined in 'MWBrowse.h'. These are:

- [BROWSE\\_EARLIEST\\_COMPATIBLE\\_VERSION](#)
- [BROWSE\\_HEADER](#)
- [BROWSE\\_VERSION](#)
- [EAccess](#)
- [EBrowserItem](#)
- [ELanguage](#)
- [EMember](#)
- [ETemplateType](#)

---

**NOTE** MWBrowse.h includes definitions for an older browser data stream format. This format is no longer used or supported by the IDE. Only current constants and enumerations are described in this section.

---

### **BROWSE\_EARLIEST\_COMPATIBLE\_VERSION**

**Description** Specifies the earliest version of the browser stream format this data is compatible with.

**Prototype** #include <MWBrowse.h>

**Remarks** A compiler stores this value in the earliest\_compatible\_version field in the [BrowseHeader](#) data structure. Usually, the value of this define matches the value of [BROWSE\\_VERSION](#).

If the browser stream format is modified in the future in a way that is compatible with earlier IDE versions, the plug-in API headers will set this value to a number less than the current browser data format version, permitting use of the plug-in's browser data with older IDE versions.

## BROWSE\_HEADER

Description	An arbitrary 64-bit signature, marking the start of a browse data stream, used by the IDE to ensure the validity of browser data.
Prototype	#include <MWBrowse.h>
Remarks	A compiler stores this value used in the <code>browser_header</code> field in the <a href="#">BrowseHeader</a> data structure.

## BROWSE VERSION

Description	Specifies the version of the browser data at compile time.
Prototype	#include <MWBrowse.h>
Remarks	<p>This predefined symbol specifies the current version of the browser API. At compile time use this symbol to determine which data structures and constants are available at compile time.</p> <p>Also, store this value in the <code>browse_version</code> field of a <a href="#">BrowseHeader</a> data structure.</p>

**EAccess**

Description	Specifies the types of access to a class member.
Prototype	#include <MWBrowse.h>
Values	The enumerated values in EAccess are:
accessNone	The item isn't accessible. Very rarely used by plug-ins.
accessPrivate	The item is accessible only from other members in the same class, but not in its descendants.
accessProtected	The item is accessible from other members in the same class and from members in the class' descendants.

accessPublic	The item is accessible from within the class and outside the class.
accessAll	The item may be accessed in any way (private, protected, and public). Typically used as a mask for the bits specifying member access, not as a valid access value.
Remarks	The EAccess enumeration lists the types of access that are available for a class data member or method. Although defined as orthogonal flags, these should not be combined.
See Also	<a href="#">“Additional Fields For Classes” on page 580</a>

## **EButtonItem**

Description	Specifies the type of item described in a browser record.
Prototype	#include <MWBrowse.h>
Values	The enumerated values in EButtonItem are:
browseFunction	The browser record describes a function, procedure, or class method.
browseGlobal	The browser record describes a global variable.
browseClass	The browser record describes a class, struct, union, or Java interface.
browseMacro	The browser record describes a macro.
browseEnum	The browser record describes an enumerated type.
browseTypedef	The browser record describes a user-defined type other than a class, struct, union, or Java interface (usually a simple scalar type, a pointer type, or an array type).
browseConstant	The browser record describes a constant definition.

browseTemplate	The browser record describes a C++ template.
browsePackage	The browser record describes a Java package.
browseCompSymbolStart	Use this as the first value to describe a compiler-specific browser item in a browser record. Subsequent types of compiler-specific browser items should use the values browseCompSymbolStart+1, browseCompSymbolStart+2, and so on. See <a href="#">“Compiler Browser Symbol Entry Point”</a> for more information on generating compiler-specific browser symbols. Records for custom compiler-specific symbol types contain no additional fields beyond those described in <a href="#">Fields For All Records</a> .
browseEnd	Use this value to mark the end of the list of browser records.
Remarks	The EBrowserItem enumeration lists the kinds of items that can be described in a browser record. All browser records start with an EBrowserItem value, which specifies the layout of the data that follows.
See Also	<a href="#">“Browser Records” on page 573</a> <a href="#">“Compiler Browser Symbol Entry Point” on page 209</a>

## **ELanguage**

Description	Used in the browse data header to specify source language type.
Prototype	#include <MWLangDefs.h>
Values	The enumerated values in ELanguage are:

langUnknown	Use langUnknown when there is no other more appropriate value in the ELanguage enumeration.
langC	Specifies C source code.
langCPlus	Specifies C++ source code.
langPascal	Specifies Pascal source code.
langObjectPasca l	Specifies Object Pascal source code.
langJava	Specifies Java source code.
langAssembler	Specifies assembly language source code (regardless of processor type).
langFortran	Specifies FORTRAN source code.
langRez	Specifies a textual source file for resources, such as a Mac OS Rez file.
Remarks	The ELanguage enumeration lists the languages that the CodeWarrior IDE browser supports. Use a value from ELanguage in the browse_language field of a <a href="#">BrowseHeader</a> structure.  If the IDE does not directly support the language of a compiler, it is permissible to use the nearest match.

## EMember

Description	Describes the type of a class member.
Prototype	#include <MWBrowse.h> typedef unsigned char EMember;
Values	The enumerated values in EAccess are:
memberFunction	The member is a routine or method.
memberData	The member is a data field.
memberEnd	A value used to mark the end of a member list, rather than to specify the type of a member record.

## Browser Reference

### *ETemplateType*

---

Remarks	The <code>EMember</code> enumeration specifies the type of a class member. This enumeration is used when describing records in a member list for a base class.
See Also	<a href="#">“member list” on page 582</a>

## **ETemplateType**

Description Describes a C++ template type.

Prototype `#include <MWBrowse.h>`  
`typedef enum ETemplateType`

Values The enumerated values in `ETemplateType` are:

- |                               |                                    |
|-------------------------------|------------------------------------|
| <code>templateClass</code>    | The template describes a class     |
| <code>templateFunction</code> | The template describes a function. |

Remarks The `ETemplateType` enumeration lists the types of C++ templates. Browser records for templates include a single byte of template-specific information containing a value of type `ETemplateType`, which specifies the template type (class or function).

See Also [“Additional Field For Templates” on page 584](#)

# PowerPC Object Code (Mac OS)

---

This chapter describes the object code format used by the CodeWarrior IDE-hosted Metrowerks plug-in compilers and linkers for PowerPC-family processors. The MPW-hosted compiler MWCPPC generates a library file instead.

Note that the Metrowerks PowerPC linker handles object code in the Metrowerks PowerPC object code format and in XCOFF format.

## PowerPC Object Code Overview

All values are in big-endian order. A `SInt32` is a 32-bit signed integral type (commonly a C/C++ `long`), while a `SInt16` is a 16-bit signed integral type (commonly a C/C++ `short`).

Note that the Metrowerks PowerPC linker handles object code in the Metrowerks PowerPC object code format and in XCOFF.

The sections in this chapter are:

- [“PowerPC Object Code Structure” on page 594](#)—describes the overall structure of a piece of PowerPC object code.
- [“PowerPC Object Header” on page 594](#)—describes the beginning of a piece of object code.
- [“PowerPC Object Data Section” on page 597](#)—describes the parts that contain the executable code and data.
- [“PowerPC Symbolic Data Header” on page 610](#)—describes the beginning of the part of object code that contains debugging information.

- [“PowerPC Symbolic Function Data Section” on page 611](#)—describes the parts of object code that contain debugging information about the routines in the object code.
- [“PowerPC Symbolic Type Data Section” on page 614](#)—describes the parts of object code that contain debugging information about the data types in the object code.
- [“PowerPC Name Table Section” on page 625](#)—describes the part of the object code that lists the symbolic names of the items in the object code.

## PowerPC Object Code Structure

The structure of CodeWarrior’s PowerPC object code can be broken down into the following sections.

- [PowerPC Object Header](#)
- [PowerPC Object Data Section](#)
- [PowerPC Symbolic Data Header](#) (optional)
  - [PowerPC Symbolic Function Data Section](#)
  - [PowerPC Symbolic Type Data Section](#)
- [PowerPC Name Table Section](#)

## PowerPC Object Header

The following describes the object header and its fields.

**Listing 14.1 PowerPC object header structure**

---

```
typedef struct ObjHeader { // object file header
    SInt32 magic word;
    SInt16 version;
    SInt16 flags;
    SInt32 obj size;
    SInt32 nametable offset;
    SInt32 nametable names;
    SInt32 syntable offset;
    SInt32 syntable size;
    SInt32 code size;
    SInt32 udata size;
```

---

```

SInt32 idata_size;
SInt32 toc_size;
SInt32 old_def_version;
SInt32 old_imp_version;
SInt32 current_version;
SInt32 reserved[13];
} ObjHeader;

```

---

**Table 14.1 PowerPC object header fields**

<b>This field</b>	<b>Contains this information</b>
magic_word	Always OBJ_MAGIC_WORD, defined to be 0x504F5752, ('POWR' in ASCII).
version	The version number of the object code format. Its value is OBJ_VERSION.
flags	A bitfield describing the type of object data, shown in <a href="#">Listing 14.2</a> .
obj_size	The size, in bytes, of the object data section that follows the object header.
nametable_offset	The offset to the first byte of the name table, relative to the start of the object header. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
nametable_names	The number of strings defined in the name table.
symtable_offset	The offset to the first byte of the symbolic data header, relative to the start of the object header. If there is no symbolic data for this object code, then this field contains 0L.

## PowerPC Object Code (Mac OS)

### PowerPC Object Header

---

This field	Contains this information
symtable_size	The total size in bytes of all of the symbolic data, including the symbolic data header, the symbolic function data section, and the symbolic type data section. If there is no symbolic data for this object code, then this field contains 0L.
code_size	The size, in bytes, of the code defined in the object data section.
udata_size	The size, in bytes, of the uninitialized data defined in the object data section.
idata_size	The size, in bytes, of the initialized data defined in the object data section.
toc_size	The size, in bytes, of the TOC data defined in the object data section.
old_def_version	The old definition version number of the code fragment. For object code that doesn't define a shared library, this field contains 0L.
old_imp_version	The old implementation version number of the code fragment. For object code that doesn't define a shared library, this field contains 0L.
current_version	The current version number of the code fragment. For object code that doesn't define a shared library, this field contains 0L.
reserved	Reserved by Metrowerks. All elements must contain 0L.

#### **Listing 14.2    Defined values for [flags](#)**

---

```
enum {
    fObjIsSharedLib = 0x0001, /* A shared library */
    fObjIsLibrary = 0x0002, /* A library */
    fObjIsPascal = 0x0004, /* A pascal source file */
```

---

```
fObjIsWeak = 0x0008,      /* A CFM lib, "Weak Import" */
fObjIsInitBefore = 0x0010 /* A CFM lib, "Initialize Before" */
};
```

---

## PowerPC Object Data Section

The object data section is composed of a series of containers called "hunks". There are several different types of hunks, each one playing a different role in the object data definition. The object data section begins with a special starting hunk, and terminates with a special ending hunk. Each hunk structure begins with a tag that uniquely identifies its type. The following are the currently-defined hunk types.

**Listing 14.3 PowerPC hunk types**

---

```
enum {
    HUNK_START=0x4567,
    HUNK_END,
    HUNK_SEGMENT,
    HUNK_LOCAL_CODE,
    HUNK_GLOBAL_CODE,
    HUNK_LOCAL_UDATA,
    HUNK_GLOBAL_UDATA,
    HUNK_LOCAL_IDATA,
    HUNK_GLOBAL_IDATA,
    HUNK_GLOBAL_ENTRY,
    HUNK_LOCAL_ENTRY,
    HUNK_IMPORT,
    HUNK_XREF_16BIT,
    HUNK_XREF_16BIT_IL,
    HUNK_XREF_24BIT,
    HUNK_XREF_32BIT,
    HUNK_XREF_32BIT_REL,
    HUNK_DEINIT_CODE,           /* Reserved */
    HUNK_LIBRARY_BREAK,         /* Obsolete */
    HUNK_IMPORT_CONTAINER,
    HUNK_SOURCE_BREAK,
    HUNK_XREF_16BIT_REL,
    HUNK_METHOD_REF,
```

---

```
HUNK_CLASS_DEF,  
HUNK_FORCE_ACTIVE  
};
```

---

The rest of this section discusses:

- [Preventing Dead-Stripping](#)
- [PowerPC Simple Hunks](#)
- [PowerPC Regular Code Hunks](#)
- [PowerPC Data Hunks](#)
- [PowerPC Alternate Entry Point Hunks](#)
- [PowerPC Cross-Reference Hunks](#)
- [PowerPC PEF Import Hunks](#)
- [PowerPC Source File Specification Hunks](#)
- [PowerPC Reserved Hunks](#)

## Preventing Dead-Stripping

A HUNK\_FORCE\_ACTIVE hunk causes the linker to never strip out the object defined by the following hunk.

## PowerPC Simple Hunks

The first hunk of the object code is of type HUNK\_START. The last hunk is of type HUNK\_END. The object code contains no other hunks of these types. The structure of these hunks is an ObjMiscHunk.

### **Listing 14.4 PowerPC simple hunk structure**

---

```
typedef struct ObjMiscHunk {  
    SInt16 hunk_type;  
    SInt16 unused;                                /* Padding */  
} ObjMiscHunk;
```

---

## PowerPC Regular Code Hunks

A HUNK\_LOCAL\_CODE hunk defines the code for a function with static (i.e. internal) linkage. Its structure is an ObjCodeHunk, followed immediately by the machine code.

A HUNK\_GLOBAL\_CODE hunk defines the code for a function with external linkage. Its structure is also an ObjCodeHunk, followed immediately by the machine code.

**Listing 14.5 PowerPC regular object code structure**

---

```
typedef struct ObjCodeHunk {
    SInt16 hunk_type;
    char sm_class;
    unsigned char
        //      multi_def : 1,
        //      over_load : 1,
        //      exported : 1,
        //      reserved : 1,          /* Reserved */
        //      alignment : 4;
    SInt32 name_id;
    SInt32 size;
    SInt32 sym_offset;
    SInt32 sym_decl_offset;
    // char code[size];
} ObjCodeHunk;
```

---

**Table 14.2 PowerPC regular object code fields**

This field	Contains this information
<u>hunk_type</u>	Either HUNK_LOCAL_CODE or HUNK_GLOBAL_CODE
<u>sm_class</u>	What type of data the hunk defines. <u><a href="#">Listing 14.6</a></u> shows the possible values for this field. The Metrowerks PowerPC linker only supports XMC_PR and XMC_GL.

---

<b>This field</b>	<b>Contains this information</b>
multi_def	If true, then this hunk may have multiple identical definitions. If false, the module does not have multiple definitions.
over_load	If true, this hunk may be overloaded by another definition. If false, the module is not overloaded by another definition.
exported	If true, this hunk will be exported. If false, the module will not be exported.
reserved	Reserved for future use.
alignment	If the value of this field is 1, then alignment is to the next byte; if 2 then alignment half-word; if 4, word; if 8, double-word; If odd,  $1 << (\text{number} >> 1)$
name_id	The item in the name table that specifies the name of the module or routine this hunk contains. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
size	The size, in bytes, of this object code hunk.
sym_offset	The byte offset of the function’s symbolic data in the function symbolic data section, relative to the start of the symbolic data header. If there is no symbolic data for this function, this field contains 0x80000000.
sym_decl_offset	The character offset where the routine is defined in the source file. If there is no symbolic data for this function, this field contains 0L.

---

### **Listing 14.6 PowerPC types for storage mapping classes**

---

```

/* Read-only classes */
#define XMC_PR 0           /* Program Code */
#define XMC_RO 1           /* Read Only Constant */
#define XMC_GL 6           /* Global Linkage */

/* Read/write classes */
#define XMC_RW 5           /* Read Write Data */
#define XMC_TC0 15          /* TOC Anchor */
#define XMC_TC 3            /* General TOC Entry */
#define XMC_TD 16          /* Scalar TOC Data */
#define XMC_DS 10          /* Routine Descriptor */

```

---

## **PowerPC Data Hunks**

A HUNK\_LOCAL\_UDATA or HUNK\_GLOBAL\_UDATA hunk defines a block of memory used for the storage of a piece of uninitialized data with static or extern linkage, respectively. Its structure is an ObjDataHunk.

A HUNK\_LOCAL\_IDATA or HUNK\_GLOBAL\_IDATA hunk defines a block of memory used for the storage of a piece of initialized data with static or extern linkage, respectively. It also specifies the data with which the memory will be initialized. Its structure is an ObjDataHunk, followed immediately by the initialized data.

### **Listing 14.7 Structure for PowerPC initialized and uninitialized data**

---

```

typedef struct ObjDataHunk {
    SInt16 hunk_type;
    char sm_class;
    unsigned char
        // multi_def : 1,
        // over_load : 1,
        // exported : 1,
        // reserved : 1,           /* Reserved */
        // alignment : 4;
    SInt32 name_id;
    SInt32 size;
    SInt32 sym_type_id;
    SInt32 sym_decl_offset;

```

---

## PowerPC Object Code (Mac OS)

### PowerPC Data Hunks

---

```
// char data[size];
} ObjDataHunk;
```

---

**Table 14.3 PowerPC initialized and uninitialized data fields**

This field	Contains this information
hunk_type	HUNK_LOCAL_IDATA, HUNK_GLOBAL_IDATA, HUNK_LOCAL_UDATA, HUNK_GLOBAL_UDATA
sm_class	XMC_RO for read-only data, XMC_RW for regular read-write data, XMC_DS for descriptor records, XMC_TC for TOC entries, XMC_TD for read-write data stored directly in the TOC, and XMC_TC0 for the TOC anchor.
	For a list of possible values, see <a href="#">Listing 14.6</a> .
multi_def	If true, then this hunk may have multiple identical definitions. If false, the module does not have multiple definitions.
over_load	If true, this hunk may be overloaded by another definition. If false, the module is not overloaded by another definition.
exported	If true, this hunk will be exported. If false, the module will not be exported.
reserved	Reserved for future use.
alignment	If the value of this field is 1, then alignment is to the next byte; if 2 then alignment half-word; if 4, word; if 8, double-word; If odd,
	$1 << (\text{number} >> 1)$

<b>This field</b>	<b>Contains this information</b>
name_id	The item in the name table that specifies the name of the item of data this hunk contains. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
size	The number of bytes that the data occupies in memory. For HUNK_LOCAL_IDATA and HUNK_GLOBAL_IDATA hunks, it is the number of bytes of initialized data which follows the ObjDataHunk.
sym_type_id	Type identifier for the data object. If the object has no data type, this field contains 0x80000000. See <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> for information on type identifiers.
sym_decl_offset	The character offset where the data is declared in the source file. If there is no symbolic data for this function, this field contains 0L.

## PowerPC Alternate Entry Point Hunks

A HUNK\_GLOBAL\_ENTRY or HUNK\_LOCAL\_ENTRY hunk defines an alternate entry point for the function that was last defined with a HUNK\_GLOBAL\_CODE or HUNK\_LOCAL\_CODE hunk. The linkage of the entry point is external with HUNK\_GLOBAL\_ENTRY and static with HUNK\_LOCAL\_ENTRY. Its structure is an ObjEntryHunk.

### **Listing 14.8 PowerPC alternate entry point for PowerPC**

---

```
typedef struct ObjEntryHunk {
    SInt16 hunk\_type;
    SInt16 unused;
    SInt32 name\_id;
    SInt32 offset;
    SInt32 sym\_type\_id;
```

---

```
SInt32 sym_decl_offset;  
} ObjEntryHunk;
```

---

**Table 14.4 PowerPC alternate entry point fields**

<b>This field</b>	<b>Contains this information</b>
hunk_type	Either HUNK_GLOBAL_ENTRY or HUNK_LOCAL_ENTRY.
unused	Padding.
name_id	The item in the name table that specifies the name of the alternate entry point. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
offset	The offset of the alternate entry point from the beginning of the module or routine contained in the previous hunk. This value is in bytes.
sym_type_id	Type identifier for the data object. If the object has no data type, this field contains 0x80000000. See <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> for information on type identifiers.
sym_decl_offset	The character offset where the routine is defined in the source file. If there is no symbolic data for this function, this field contains 0L.

---

## PowerPC Cross-Reference Hunks

Hunks of types HUNK\_XREF\_16BIT, HUNK\_XREF\_16BIT\_IL, HUNK\_XREF\_24BIT, HUNK\_XREF\_32BIT, HUNK\_XREF\_32BIT\_REL, and HUNK\_XREF\_16BIT\_REL mark positions in the previous code or data hunk whose value depends on the load address of another symbol. The different types are for the different relocation addressing modes supported by the linker. These hunks must follow a code hunk and any entry hunks it may have. Each of their structures is an ObjXRefHunk.

---

**Listing 14.9 PowerPC cross-reference structure**

---

```
typedef struct ObjXRefHunk {
    SInt16 hunk_type;
    char sm_class;
    char unused;
    SInt32 name_id;
    SInt32 offset;
} ObjXRefHunk;
```

---

**Table 14.5 PowerPC cross-reference fields**

This field	Contains this information
hunk_type	HUNK_XREF_16BIT, HUNK_XREF_16BIT_IL, HUNK_XREF_24BIT, HUNK_XREF_32BIT, HUNK_XREF_32BIT_REL, HUNK_XREF_16BIT_REL
sm_class	The storage-mapping class of referenced symbol. See <a href="#">Listing 14.6</a> for information on storage mapping classes.
unused	Padding.
name_id	The item in the name table that specifies the name of the hunk being referred to. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
offset	The offset of the cross-reference within the hunk that this hunk applies to.

---

## PowerPC PEF Import Hunks

A HUNK\_IMPORT\_CONTAINER hunk specifies the name and PEF version numbers for a code fragment from which symbol definitions will be imported. Hunks of this type only appear in shared library object code. Its structure is ObjContainerHunk.

## PowerPC Object Code (Mac OS)

### PowerPC PEF Import Hunks

---

#### **Listing 14.10 PowerPC PEF import container structure**

---

```
typedef struct ObjContainerHunk {
    SInt16 hunk_type;
    UInt16 flags;
        // unused: 15,      /* Reserved */
        // auto_weak: 1;
    SInt32 name_id;
    SInt32 old_def_version;
    SInt32 old_imp_version;
    SInt32 current_version;
} ObjContainerHunk;
```

---

**Table 14.6 PowerPC PEF import container fields**

This field	Contains this information
hunk_type	HUNK_IMPORT_CONTAINER
flags auto_weak	This item is 1 if the linker should automatically mark the imported fragment as weak regardless of the user's "Import Weak" setting. The PEF Importer sets this bit if the shared library's cfrg resource's usage field equals kWeakStubLibraryCFrag (4).
name_id	The item in the name table that specifies the run-time name of the import container. For more information on the name table see <a href="#">"PowerPC Name Table Section" on page 625</a> .
old_def_version	The old definition version of this PEF container.
old_imp_version	The old implementation version of this PEF container.
current_version	The current version of this PEF container.

---

A HUNK\_IMPORT hunk specifies a symbol that will be imported from a different code fragment (the one that was last named with a

HUNK\_IMPORT\_CONTAINER hunk). A hunk of this type only appears in the object data for a shared library. Its structure is an ObjImportHunk.

#### **Listing 14.11 PowerPC PEF import symbol structure**

---

```
typedef struct ObjImportHunk {
    SInt16 hunk_type;
    char sm_class;
    char unused;
    SInt32 name_id;
} ObjImportHunk;
```

---

**Table 14.7 PowerPC PEF import fields**

<b>This field</b>	<b>Contains this information</b>
hunk_type	HUNK_IMPORT
sm_class	The storage class of the imported symbol: XMC_RW or XMC_DS. See <a href="#">Listing 14.6</a> for other information on storage classes.
unused	Padding.
name_id	The item in the name table that specifies the name of the imported symbol. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .

---

## **PowerPC Source File Specification Hunks**

A HUNK\_SOURCE\_BREAK hunk specifies the name of the source file which defines subsequent code and data. The hunk contains the index into the name table which contains the file's full path. This hunk can be used to tell a source-level debugger, for example, that the following code and data are actually defined in a header file. If the name index of the source file in a subsequent HUNK\_SOURCE\_BREAK hunk is 0, then the source file reverts to the original file. Its structure is ObjSourceHunk.

## **PowerPC Object Code (Mac OS)**

### *PowerPC Object Pascal Hunks*

---

#### **Listing 14.12    Source file specification structure for PowerPC**

---

```
typedef struct ObjSourceHunk {
    SInt16 hunk_type;
    SInt16 unused;
    SInt32 name_id;
    SInt32 moddate;
} ObjSourceHunk;
```

---

**Table 14.8    PowerPC source file specification fields**

<b>This field</b>	<b>Contains this information</b>
hunk_type	HUNK_SRC_BREAK
unused	Padding.
name_id	The item within the name table that specifies the full path name of the file that contains the source code. Use 0L to specify the original source file. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
moddate	The modification date of this file, in the format used by the Mac OS GetDate() routine.

---

## **PowerPC Object Pascal Hunks**

Metrowerks Object Pascal uses the HUNK\_METHOD\_REF hunk type to specify a method table.

---

#### **Listing 14.13    PowerPC Object Pascal method table structure**

---

```
typedef struct ObjMethHunk {
    SInt16 hunk_type;
    SInt32 name_id;
    SInt32 size;
} ObjMethHunk;
```

---

**Table 14.9 PowerPC Object Pascal method table fields**

This field	Contains this information
hunk_type	HUNK_METHOD_REF
name_id	The item within the name table that specifies the name of the method table. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625.</a>
size	The number of method table references.

Object Pascal uses the HUNK\_CLASS\_DEF hunk type to specify to specify a class.

**Listing 14.14 PowerPC Object Pascal class structure**

---

```
typedef struct ObjClassHunk {
    SInt16 hunk_type;
    SInt32 name_id;
    SInt16 methods;
    SInt16 pairs;
} ObjClassHunk;
```

---

**Table 14.10 PowerPC Object Pascal class fields**

This field	Contains this information
hunk_type	HUNK_CLASS_REF
name_id	The item within the name table that specifies the name of the class. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625.</a>
methods	The number of methods in the class.
pairs	The number of base classes. See <a href="#">Listing 14.15</a> for information on base class pairs.

## **PowerPC Object Code (Mac OS)**

### *PowerPC Reserved Hunks*

Class base pairs are specified in an `ObjClassPair` data structure:

#### **Listing 14.15 PowerPC Object Pascal class base pair**

---

```
typedef struct ObjClassPair {
    SInt32 base_id;
    SInt32 bias;
} ObjClassPair;
```

---

#### **Table 14.11 PowerPC Object Pascal class base pair fields**

<b>This field</b>	<b>Contains this information</b>
base_id	The base class ID within the name table. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625.</a>
bias	The base bias in the class.

---

## **PowerPC Reserved Hunks**

Hunks of type HUNK\_SEGMENT are not used in PowerPC object code.

The HUNK\_INIT\_CODE, HUNK\_deinit\_CODE and HUNK\_LIBRARY\_BREAK hunk types are reserved and should not be used.

## **PowerPC Symbolic Data Header**

The symbolic data header, the symbolic function data section, and the symbolic type data section are optional. The location of the symbolic data header is recorded in the object header. The following describes the symbolic data header and its fields.

#### **Listing 14.16 PowerPC symbolic data header structure**

---

```
typedef struct {
    SInt32 header_id;
    SInt32 typeoffset;
```

---

```
SInt32 types;  

SInt32 unnamed;  

SInt32 res[4];  

} SymHeader;
```

---

**Table 14.12 PowerPC symbolic data header fields**

This field	Contains this information
header_id	Always SYM_HEADER_ID, defined to be 0x53594D48 ('SYMH' in ASCII).
typeoffset	The byte offset to the type data, relative to the start of the symbolic data header.
types	The number of types defined in the type data section.
unnamed	The number of unnamed types defined in the type data section. Structure and enumerated types are unnamed in C/C++ when they don't have a tag. Anonymous types are also unnamed in Pascal.
res	Reserved for future use. This is an array of 4 SInt32 values, each containing 0L.

---

## PowerPC Symbolic Function Data Section

The symbolic function data section is a series of variable-length records, each containing information describing a function or procedure. Each record begins with a SInt16 that contains the value 0 for a procedure (that is, a routine without a return value) and 1 for a function (a routine that returns a value).

The SInt16 is followed by a list of statement locations. Each statement location creates a correspondence between an offset in the function code and a location in the source code where the user can place a breakpoint in a debugger. Each entry in the statement list is composed of two SInt32 values. The first is the code offset, relative to the start of the function. The second SInt32 value is the character offset of the statement location in the source file. The list is

terminated by a special list entry whose first SInt32 is 0xFFFF and whose second SInt32 is the same as the last statement's.

The statement list is followed by a SInt16 containing the number of local variables and parameters defined in this function. This is immediately followed by that number of local variable definitions, each with the following structure:

**Listing 14.17 PowerPC local variable and parameter structure**

---

```
typedef struct {
    SInt32  name;
    SInt32  type;
    char    kind;
    char    sclass;
    SInt32  where;
} LocalVar;
```

---

**Table 14.13 PowerPC local variable and parameter fields**

This field	Contains this information
name	The index of the local variable's name in the name table section. See " <a href="#">“PowerPC Name Table Section” on page 625</a> " for more information on the name table.
type	The type identifier of the local variable. For information on type identifiers, see " <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> ".
kind	How the variable is stored. This field contains one of the values listed in <a href="#">Listing 14.18</a> .
sclass	The variable's storage class. This field contains one of the values listed in <a href="#">Listing 14.19</a> .
where	Where the variable is stored. This value depends on the value of the <a href="#"><u>sclass</u></a> field. See <a href="#">Listing 14.19</a> for more information.

---

The value of the kind field is one of the following values:

**Listing 14.18 Kinds of local variable and parameter storage for PowerPC**

```
enum {
    STORAGE_KIND_LOCAL=0,
    STORAGE_KIND_VALUE,
    STORAGE_KIND_REFERENCE
};
```

---

**Table 14.14 PowerPC local variable and parameter storage values**

<b>This value</b>	<b>Specifies this information</b>
STORAGE_KIND_LOCAL	Local variables.
STORAGE_KIND_VALUE	Parameters.
STORAGE_KIND_REFERENCE	Pascal VAR parameters.

**Listing 14.19 Register storage for PowerPC**

```
enum {
    STORAGE_CLASS_REGISTER=0,
    STORAGE_CLASS_A5,
    STORAGE_CLASS_A6,
    STORAGE_CLASS_A7
};
```

---

**Table 14.15 PowerPC register storage values**

<b>This value</b>	<b>Specifies this information</b>
STORAGE_CLASS_REGISTER	A variable stored in a register. The <a href="#">where</a> field identifies the register: 0-31 are used for GPRs, 32-63 are used for FPRs.
STORAGE_CLASS_A6	a variable stored on the stack in a function that supports <code>alloca()</code> . GPR31 is reserved as a stack frame pointer.
STORAGE_CLASS_A7	The <a href="#">where</a> field contains the stack frame offset.
STORAGE_CLASS_A7	A variable stored on the stack.
STORAGE_CLASS_A5	The <a href="#">where</a> field contains the stack offset.
STORAGE_CLASS_A5	Not used.

## PowerPC Symbolic Type Data Section

Certain types are predefined and don't require type definitions. The following are the predefined types and their identifier values:

**Listing 14.20 PowerPC symbolic type data section structure**

```
enum {
    BASICTYPE_VOID,                      /* No type */
    BASICTYPE_PSTRING,                   /* Pascal string */
    BASICTYPE_ULONG,                     /* unsigned long word */
    BASICTYPE_LONG,                      /* signed long word */
    BASICTYPE_FLOAT10,                  /* extended (10 bytes) */
    BASICTYPE_BOOLEAN, /* Pascal boolean (1 byte) */
    BASICTYPE_UBYTE,                     /* unsigned byte */
    BASICTYPE_BYTE,                      /* signed byte */
    BASICTYPE_CHAR,                      /* character (1 byte) */
    BASICTYPE_WCHAR,                    /* character (2 bytes) */
};
```

```
BASICTYPE_UWORD,           /* unsigned word */
BASICTYPE_WORD,            /* signed word */
BASICTYPE_FLOAT4,          /* single */
BASICTYPE_FLOAT8,          /* double */
BASICTYPE_FLOAT12,         /* extended (12 bytes) */
BASICTYPE_COMP,             /* computational (8 bytes) */
BASICTYPE_CSTRING,          /* C string */
BASICTYPE_AISTRING,        /* as-is string */
MYBASICTYPE_VOIDPTR=100,    /* void */
MYBASICTYPE_VOIDHDL,       /* void ** */
MYBASICTYPE_CHARPTR,        /* char */
MYBASICTYPE_CHARHDL,       /* char ** */
MYBASICTYPE_UCHARPTR,      /* unsigned char */
MYBASICTYPE_UCHARHDL,      /* unsigned char ** */
MYBASICTYPE_FUNC,           /* void func(void) */
MYBASICTYPE_STRINGPTR,     /* C string pointer */
MYBASICTYPE_PSTRINGPTR,    /* Pascal str. pointer */
};


```

---

The MYBASICTYPE\_CHARPTR type differs from the MYBASICTYPE\_STRINGPTR type only in semantics. A debugger might display a MYBASICTYPE\_CHARPTR type as a pointer to character, while displaying a MYBASICTYPE\_STRINGPTR type directly as a C string without any dereferencing or type coercion.

The rest of this section discusses the following types:

- [Other Types for PowerPC](#)
- [PowerPC Pointer Type](#)
- [PowerPC Array Type](#)
- [PowerPC Structured Type](#)
- [PowerPC Enumerated Type](#)
- [PowerPC Pascal Array Type](#)
- [PowerPC Pascal Subrange Type](#)
- [PowerPC Pascal String Type](#)

## **Other Types for PowerPC**

Other types require a type definition. The symbolic type data section stores these type definitions in a series of variable-length

records. The number of user-defined types is stored in the symbolic data header.

Each type definition begins with a tag and a type ID. The `SInt16` tag identifies the structure of the remainder of the definition. It can have one of the following values:

The `SInt32` type ID is unique identifier of the type being defined. Type IDs for user-defined types are sequentially defined in decreasing order, starting with -1. Positive type IDs are reserved for predefined types.

**Listing 14.21 Other data types for PowerPC**

```
enum {
    LOCTYPE_POINTER,
    LOCTYPE_ARRAY,
    LOCTYPE_STRUCT,
    LOCTYPE_ENUM,
    LOCTYPE_PARRAY,
    LOCTYPE_RANGE,
    LOCTYPE_SET,
    LOCTYPE_PENUM,
    LOCTYPE_PSTRING
};
```

**Table 14.16 PowerPC other data type tags**

This tag	Specifies this information
LOCTYPE_POINTER	A pointer type. See <a href="#">“PowerPC Pointer Type” on page 617</a> .
LOCTYPE_ARRAY	An array type. See <a href="#">“PowerPC Array Type” on page 618</a> .
LOCTYPE_STRUCT	A structured type. See <a href="#">“PowerPC Structured Type” on page 618</a> .
LOCTYPE_ENUM	An enumeration type. See <a href="#">“PowerPC Enumerated Type” on page 619</a> .

This tag	Specifies this information
LOCTYPE_PARRAY	A Pascal array type. See <a href="#">“PowerPC Pascal Array Type” on page 621</a> .
LOCTYPE_RANGE	A Pascal subrange type. See <a href="#">“PowerPC Pascal Subrange Type” on page 622</a> .
LOCTYPE_SET	A Pascal set type. See <a href="#">“PowerPC Pascal Set Type” on page 623</a> .
LOCTYPE_PENUM	A Pascal enumeration type. See <a href="#">“PowerPC Pascal Enumerated Type” on page 623</a> .
LOCTYPE_PSTRING	A Pascal string type. See <a href="#">“PowerPC Pascal String Type” on page 624</a> .

## PowerPC Pointer Type

A pointer type definition consists of a LOCTYPE\_POINTER tag, followed by the SInt32 type ID being defined, followed by a LocPointerStruct.

**Listing 14.22 PowerPC pointer data type structure**

---

```
typedef struct LocPointerStruct {
    SInt16 number;
    SInt32 type;
} LocPointerStruct;
```

---

**Table 14.17 PowerPC pointer type fields**

This field	Contains this information
number	The number of pointer indirections. For example, in C/C++, 1 is equivalent to *, 2 is **, 3 is *** and so on.
type	The type ID of the item being pointed to. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .

## PowerPC Array Type

An array type definition consists of a LOCTYPE\_ARRAY tag, followed by the SInt32 type ID being defined, followed by a LocArrayStruct.

**Listing 14.23 PowerPC array data type structure**

---

```
typedef struct LocArrayStruct {  
    SInt32 size;  
    SInt32 esize;  
    SInt32 type;  
} LocArrayStruct;
```

---

**Table 14.18 PowerPC array type fields**

This field	Contains this information
size	Size of the array, in bytes.
esize	Size of a single element.
type	The array elements' type ID. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .

---

## PowerPC Structured Type

A struct type definition consists of a LOCTYPE\_STRUCT tag, followed by the SInt32 type ID being defined, followed by a LocStructStruct, followed by a LocStructStructMember for each member of the struct.

**Listing 14.24 PowerPC structured data type structure**

---

```
typedef struct LocStructStruct {  
    SInt32 name;  
    SInt32 size;  
    SInt16 members;  
    // LocStructStructMember member[members];  
} LocStructStruct;
```

---

**Table 14.19 PowerPC structured type fields**

This field	Contains this information
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
size	Size of the structure.
members	The number of members within the structure.

**Listing 14.25 PowerPC structured type member structure**

---

```
typedef struct LocStructStructMember {
    SInt32 name;
    SInt32 type;
    SInt32 offset;
} LocStructStructMember;
```

---

**Table 14.20 PowerPC member structure fields**

This field	Contains this information
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
type	The type ID of the member. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .
offset	The offset of the member from the beginning of the structure, in bytes.

## PowerPC Enumerated Type

An enumerated type definition consists of a LOCTYPE\_ENUM tag, followed by the SInt32 type ID being defined, followed by a LocEnumStruct, followed by a LocEnumMember for each member of the struct.

## **PowerPC Object Code (Mac OS)**

### *PowerPC Enumerated Type*

---

#### **Listing 14.26 PowerPC enumerated type structure**

---

```
typedef struct LocEnumStruct {  
    SInt32  name;  
    SInt16  baseid;  
    SInt16  members;  
    // LocEnumMember    member[members];  
} LocEnumStruct;
```

---

**Table 14.21 PowerPC enumeration type fields**

<b>This field</b>	<b>Contains this information</b>
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
baseid	The type ID of the enumeration’s base type. This must always be a basic type. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .
members	The number of members within the enumeration.

#### **Listing 14.27 PowerPC enumerated type member structure**

---

```
typedef struct LocEnumMember {  
    SInt32  name;  
    SInt32  value;  
} LocEnumMember;
```

---

**Table 14.22 PowerPC member enumeration fields**

<b>This field</b>	<b>Contains this information</b>
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
value	The value of the enumeration member.

## PowerPC Pascal Array Type

A Pascal array type definition consists of a LOCTYPE\_PARRAY tag, followed by the SInt32 type ID being defined, followed by a LocPArrayStruct.

**Listing 14.28 PowerPC Pascal array type structure**

---

```
typedef struct LocPArrayStruct {
    SInt32 pckd;
    SInt32 size;
    SInt32 iid;
    SInt32 eid;
    SInt32 name;
} LocPArrayStruct;
```

---

**Table 14.23 PowerPC Pascal array type fields**

<b>This field</b>	<b>Contains this information</b>
pckd	True if the array is packed, false otherwise.
size	The size, in bytes, of the array.
iid	The type ID of the array's index.

## PowerPC Object Code (Mac OS)

### PowerPC Pascal Subrange Type

---

This field	Contains this information
eid	The type ID of the array's elements. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .

## PowerPC Pascal Subrange Type

A Pascal subrange type definition consists of a LOCTYPE\_RANGE tag, followed by the SInt32 type ID being defined, followed by a LocRangeStruct.

**Listing 14.29 PowerPC Pascal subrange structure**

---

```
typedef struct LocRangeStruct {  
    SInt32  name;  
    SInt32  base;  
    SInt32  size;  
    SInt32  lbound;  
    SInt32  hbound;  
} LocRangeStruct;
```

---

**Table 14.24 PowerPC Pascal subrange type fields**

This field	Contains this information
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
base	The type ID of the subrange's base type. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .
size	The size, in bytes, of the subrange.

<b>This field</b>	<b>Contains this information</b>
lbound	The lower bound value of the subrange.
hbound	The upper bound value of the subrange.

## **PowerPC Pascal Set Type**

A Pascal set type definition consists of a LOCTYPE\_SET tag, followed by the SInt32 type ID being defined, followed by a LocSetStruct.

**Listing 14.30 PowerPC Pascal set type structure**

---

```
typedef struct LocSetStruct {
    SInt32 name;
    SInt32 base;
    SInt32 size;
} LocSetStruct;
```

---

**Table 14.25 PowerPC Pascal set type fields**

<b>This field</b>	<b>Contains this information</b>
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
base	The type ID of the set’s base type. For information on type identifiers, see <a href="#">“PowerPC Symbolic Type Data Section” on page 614</a> .
size	The size, in bytes, of the set.

## **PowerPC Pascal Enumerated Type**

A Pascal enumerated type definition consists of a LOCTYPE\_PENUM tag, followed by the SInt32 type ID being defined, followed by a LOCPEnumStruct, followed by a SInt32 for each enumerated value of the type. The values of the SInt32s are the indexes into the name table for the names of the enumerated values.

## **PowerPC Object Code (Mac OS)**

### *PowerPC Pascal String Type*

**Listing 14.31 PowerPC Pascal enumerated type structure**

---

```
typedef struct LocPEnumStruct {
    SInt32 name;
    SInt32 count;
// SInt32 cname[count];
} LocPEnumStruct;
```

---

**Table 14.26 PowerPC Pascal enumeration type fields**

<b>This field</b>	<b>Contains this information</b>
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .
count	The number of constants in the enumeration.

## **PowerPC Pascal String Type**

**Listing 14.32 PowerPC Pascal string type structure**

---

```
typedef struct LocPStringStruct {
    SInt32 size;
    SInt32 name;
} LocPStringStruct;
```

---

**Table 14.27 PowerPC member enumeration fields**

<b>This field</b>	<b>Contains this information</b>
size	The number of elements in the character string array.
name	The item within the name table that specifies the name of the type. For more information on the name table see <a href="#">“PowerPC Name Table Section” on page 625</a> .

## PowerPC Name Table Section

The number of string table entries is specified in the object header. Each entry of the name table consists of a 16 bit hash value, followed by a null-terminated character string no longer than 255 characters.

Use the routine defined in [Listing 14.33](#) to compute a hash value for the PowerPC name table section.

Other structures in the object code reference strings in the name table section by a number from 1 to the total number of strings, not by a byte offset. The index -1 is special and indicates the empty name for unnamed types.

### **Listing 14.33 Getting a hash value for the PowerPC name table section**

```
#define NAMEHASH 2048

SInt16 CHash(
    char *string )
{
    SInt16 i, hashval;
    unsigned char u;

    if ( ( hashval = strlen( string ) & 0x0FF ) != 0 )
    {
        for ( i = hashval, u = 0; i > 0; i-- )
        {
            u = ( u >> 3 ) | ( u << 5 );
            u += *string++;
        }
        hashval = ( hashval << 8 ) | u;
    }
    return ( hashval & ( NAMEHASH - 1 ) );
}
```

## **PowerPC Object Code (Mac OS)**

### *PowerPC Name Table Section*

---

# Library Specification (Mac OS)

---

This chapter describes the library format used by the Metrowerks compilers and linkers for Motorola 68000-family processors and PowerPC processors.

## Library Specification Overview

The library is a file representation of one or more object code modules.

On Mac OS, a library has a file type of 'MPLF', and a creator of 'CWIE'. The data fork contains the library code. The resource fork is reserved for Metrowerks internal use.

All values are stored in big-endian order. A `SInt32` is a 32-bit signed integral type (commonly a `long`), while a `UInt32` is a 32-bit unsigned integral type (commonly an `unsigned long`).

The sections in this chapter are:

- [“Overall Structure” on page 628](#)—describes the organization of the data in a library.
- [“Library Header” on page 628](#)—describes the information stored at the beginning of a library.
- [“File Data Records” on page 629](#)—describes the entries in a library that represent information about the files that the library’s object was compiled from.
- [“File and Path Names” on page 630](#)—describes where in an what format file and path names are stored in a library.
- [“Object Code Modules” on page 630](#)—describes how and where to store the object code for a library.

# Overall Structure

The structure of CodeWarrior's library format can be broken down into the following sections.

- [Library Header](#)
- [File Data Records](#)
- [File and Path Names](#)
- [Object Code Modules](#)

## Library Header

The library header is structured as follows:

**Listing 15.1 Library header structure**

---

```
typedef struct LibHeader {  
    SInt32    magicword;  
    SInt32    magicproc;  
    SInt32    magicflags;  
    SInt32    version;  
    SInt32    code size;  
    SInt32    data size;  
    SInt32    nobjectfiles;  
    // LibFile objectfiles[nobjectfiles];  
} LibHeader;
```

---

**Table 15.1 Library header field descriptions**

This field	Contains this information
magicword	Always LIB_MAGIC_WORD, defined to be 0x4D574F42, ('MWOB' in ASCII).
magicproc	Processor type. For a PowerPC library use 0x50504320 ('PPC' in ASCII). For a 68K library use 0x4D36384B ('M68K' in ASCII).
magicflags	Reserved. Its value should be 0L.

<b>This field</b>	<b>Contains this information</b>
version	The version number of the object code format. Its value is 1 for PowerPC libraries and 2 for 68K libraries.
code_size	The total size, in bytes, of all of the executable code defined in the library.
data_size	The total size, in bytes, of all of the data defined in the library.
nobjectfiles	The number of object files encapsulated by the library. Following the library header, there will be this number of file data records, and the same number of object data modules.

## File Data Records

The file data records follow the library header. There is one record for each of the library's constituent files. The structure of each record follows:

**Listing 15.2 File data record structure**

---

```
typedef struct LibFile {
    UInt32 moddate;
    SInt32 filename;
    SInt32 fullpathname;
    SInt32 objectstart;
    SInt32 objectszie;
} LibFile;
```

---

## Library Specification (Mac OS)

### File and Path Names

---

**Table 15.2 File data record field descriptions**

This field	Contains this information
moddate	The date and time when the file was last compiled, in the number of seconds since midnight of January 1, 1904. This is the same date and time format used by the <code>GetDateTime()</code> Mac OS call.
filename	The byte offset of this file's filename, relative to the start of the library header.
fullpathname	The byte offset of this file's complete pathname, relative to the start of the library header.
objectstart	The byte offset of this file's object code module, relative to the start of the library header.
objectszie	The size in bytes of this file's object module.

## File and Path Names

The file and path names of the library's constituent files are stored after the file data records. They are null-terminated C strings.

## Object Code Modules

The object code modules come last. The object code formats are documented in [“PowerPC Object Code \(Mac OS\)” on page 593](#). The object code modules are aligned on 4-byte boundaries, so there might be up to three bytes of zero padding before each object code module. The size of this padding is not included in the [objectszie](#) in the corresponding file data record.

# External Editors on Mac OS

---

This chapter discusses the AppleEvent interface used by the CodeWarrior IDE to support third-party text editors.

## External Editors on Mac OS Overview

The CodeWarrior IDE provides an AppleEvent-based interface that an external text editor can use to connect with CodeWarrior. This provides a CodeWarrior programmer with the opportunity to use an editor other than the one included with the IDE. Interaction between the IDE and the external editor is seamless from the user's point of view.

The CodeWarrior IDE uses AppleEvents to communicate with the external editor. When necessary, the IDE sends specific AppleEvents that tell the editor to:

- open a file
- provide all the text in the file to the IDE
- identify open files for the IDE
- identify open modified files for the IDE

The editor must support all of these events. These are the only events sent to the editor by the IDE. Other functionality that exists between the IDE and its own editor is not available with external editors. For example, you cannot find text using the IDE's search engine, and display the text using the external editor.

In addition to responding to events received from the IDE, the editor must send an AppleEvent to the IDE when a file has been modified. This allows the IDE to automatically mark the file as

changed. Otherwise the user must explicitly tell the IDE to update file modification dates.

If the editor does not notify the IDE of changed files, and the programmer forgets to update file modification dates manually, the build system may not update the object code that depends upon the changed file.

In addition to notifying the IDE that a source file has changed, the external editor may send any other AppleEvent that the IDE recognizes. The external editor can use the IDE to compile, check syntax, and so forth.

The specific AppleEvents in the IDE/external editor interface are:

- [Open Document](#)
- [Get Text](#)
- [Window Search](#)
- [Modified \(from IDE to Editor\)](#)
- [Modified \(from Editor to IDE\)](#)

This chapter also covers these additional topics:

- [Sending Other Events to the IDE](#)
- [Structure Alignment for External Editors](#)
- [Using an External Editor](#)

---

**NOTE** This chapter assumes you are familiar with AppleEvent programming. If you are not, consult *Inside Macintosh* or any book on Mac OS programming that discusses AppleEvents.

---

## Open Document

Description	The IDE sends this event to the external editor to tell it to open a file.
Event Class	kCoreClassEvent
Event ID	kOpenDocuments
Parameters	The AppleEvent parameters for this event are:

Key	Type	Usage	Description
keyDirectObject	typeFSS	required	identifies the file to open
keyAEPPosition	typeChar	optional	a structure to specify the selection range

The optional structure is declared as:

---

```
struct SelectionRange
{
    short  unused1;      // 0 (not used)
    short  lineNumber;   // line to select (<0 to specify range)
    long   startRange;   // start of selection range (if line < 0)
    long   endRange;     // end of selection range (if line < 0)
    long   unused2;      // 0 (not used)
    long   theDate;      // modification date/time
};
```

---

You are free to use any names for the structure and its fields.

When declaring this structure, the IDE uses 68K alignment. The external editor must do so as well. For information on setting alignment, see [“Structure Alignment for External Editors” on page 638.](#)

---

**NOTE** The constants kCoreClassEvent, kOpenDocuments, keyDirectObject, and keyAEPPosition, are declared in Apple Computer’s universal headers for Mac OS programming.

---

Event Reply      None.

Remarks      The IDE uses the optional keyAEPPosition parameter to tell the editor the selection range. If lineNumber is zero or greater, scroll the text to the specified line. If lineNumber is less than zero, use the values in startRange and endRange to select the specified characters. Scroll the text to display the selection. If lineNumber, startRange, and endRange are all negative, there is no selection range specified.

## Get Text

Description	The IDE sends the Get Text AppleEvent to the editor when it needs the source code from a file. For example, when the user issues a Check Syntax or Compile command, the compiler needs access to the source code contained in the file.
Event Class	'KAHL'
Event ID	'GTTX'
Parameters	The AppleEvent parameter for this event is:

Key	Type	Usage	Description
keyDirectObject	typeChar	required	a structure that contains the necessary information

The structure is declared as:

---

```
struct GetText
{
    FSSpec theFile; // identifies the file
    Handle theText; // the location where you return the text
    long *unused; // 0 (not used)
    long *theDate; // where to put the modification date/time
};
```

---

You are free to use any names for the structure and its fields.

**WARNING!**

When the editor receives this event, the address in theText is a handle to a block of zero size. You must resize the handle to the correct size before storing data at that address.

---

When declaring this structure, the IDE uses 68K alignment. The external editor must do so as well. For information on setting alignment, see ["Structure Alignment for External Editors" on page 638](#).

Event Reply      None. Put data in locations specified in the structure received.

Remarks      When the editor receives this event, it must set the size of the handle in theText to fit the data in the file. It must then copy the entire

---

contents of the specified file into the memory location specified in theText.

## Window Search

Description	The IDE sends the Window Search AppleEvent to the editor when it needs to know whether a particular file is open in the editor.
Event Class	'KAHL'
Event ID	'SRCH'
Parameters	The AppleEvent parameter for this event is:

Key	Type	Usage	Description
keyDirectObject	typeChar	required	a structure that contains the necessary information

The structure is declared as:

---

```
struct WindowSearch
{
    FSSpec theFile; // identifies the file
    long *theDate; // where to put the modification date/time
};
```

---

You are free to use any names for the structure and its fields.

When declaring this structure, the IDE uses 68K alignment. The external editor must do so as well. For information on setting alignment, see [“Structure Alignment for External Editors” on page 638.](#)

Event Reply	None. Put data in the location specified in the structure received.
Remarks	When the editor receives this event, determine whether the specified file is open. If it is, return the modification date/time for that file in the appropriate location specified in the structure. If the file is not opened, put the value fnfErr (file not found) in that location.

## External Editors on Mac OS

Modified (from IDE to Editor)

---

### Modified (from IDE to Editor)

Description	The IDE sends this event to the external editor when it wants to know which files that are open in the editor have been modified.
Event Class	'KAHL'
Event ID	'MOD'
Parameters	None.
Event Reply	The reply for this event is:

Key	Type	Usage	Description
keyDirectObject	typeAEList	required	each element in the list is a structure of typeChar

The structure is declared as:

---

```
struct ModificationInfo
{
    FSSpec theFile; // identifies the file
    long theDate; // the date/time the file was last modified
    short saved; // set this to zero when replying
};
```

---

You are free to use any names for the structure and its fields.

When declaring this structure, the IDE uses 68K alignment. The external editor must do so as well. For information on setting alignment, see ["Structure Alignment for External Editors" on page 638](#).

Remarks When building the reply event, include one element in the list for each open file that has been modified.

See Also [Modified \(from Editor to IDE\)](#)

### Modified (from Editor to IDE)

Description The external editor sends this event to the IDE when a file is modified. The editor should do this when it saves a file, so that the

---

IDE's project manager can mark the file as changed for the build system.

Event Class      'KAHL'

Event ID        'MOD '

Parameters     The AppleEvent parameter for this event is:

<b>Key</b>	<b>Type</b>	<b>Usage</b>	<b>Description</b>
keyDirectObject	typeAEList	required	each element in the list is a structure of typeChar

The structure is declared as:

```
struct ModificationInfo
{
    FSSpec theFile; // identifies the file
    long theDate; // the date/time the file was last modified
    short saved; // unused
};
```

You are free to use any names for the structure and its fields.

When declaring this structure, the IDE uses 68K alignment. The external editor must do so as well. For information on setting alignment, see ["Structure Alignment for External Editors" on page 638.](#)

Event Reply     None.

Remarks        When building the reply event, include one element in the list for each file that has changed. Typically this will be one file, and should occur whenever the user saves changes. You should also notify the IDE if changes are saved when the user closes a window.

If the editor does not notify the IDE of a changed file, the build system may not update the object code that depends upon the file.

See Also      [Modified \(from IDE to Editor\)](#)

## Sending Other Events to the IDE

The CodeWarrior IDE on Mac OS is scriptable. It understands and responds to a variety of AppleEvents. Any other program, including an external editor, may send a supported AppleEvent to the IDE to tell the IDE to perform some task.

For example, an external editor might support a compile command. When the user makes a change to a source file and chooses the command *in the external editor application*, the editor can send the appropriate AppleEvent to the CodeWarrior IDE. In response, the IDE will compile the file. All of this happens transparently to the user.

For several key AppleEvents, the IDE recognizes the fact that the command may be coming from an external editor. It has the facility to return error messages to the external editor, which would be responsible for interacting with the user in response to the error. These AppleEvents include:

Check Syntax	Compile
Make Project	Precompile
Preprocess	Run Project
Update Project	

For full information on these and other AppleEvents, refer to the Mac OS version of the *IDE User Guide*.

## Structure Alignment for External Editors

All of the structures used in the AppleEvent interface between the IDE and external editors must be created using 68K alignment. Otherwise the IDE and the editor may not be able to read the data in the structures properly.

When you build the editor with CodeWarrior, you can set structure alignment in one of two ways:

- for the entire target
- for the structures only

To set a target's alignment, choose 68K alignment as the Struct Alignment option in the 68K Processor and/or the PPC Processor settings panels. For more information on these settings panels, see the *Targeting Mac OS* manual.

If you prefer to have a different default alignment for a target, you can modify the alignment before declaring the structures using

---

```
#pragma options align=mac68k
```

---

After declaring the structures, use

---

```
#pragma options align=reset
```

---

to restore alignment to the target's default value. For more information on pragmas, see the *C Compilers Reference*.

## **Using an External Editor**

The IDE will use the external editor when two conditions are met:

1. The Use External Editor option in the Extras preference panel is selected (see the *IDE User Guide* for more information).
2. An alias named `External Editor` is located in the `(Helper Apps)` folder. The folder is located in the same directory as the CodeWarrior IDE application. The alias should point to the actual location of the external editor.

## **External Editors on Mac OS**

*Using an External Editor*

---

# Mac OS Plug-in Resource Formats

---

This chapter documents the resource formats optionally used by the Macintosh IDE plug-ins to describe plug-in capabilities to the IDE.

## Resources for Compiler and Linker Plug-ins (Mac OS)

The resources documented in this chapter describe a plug-in and its capabilities to the IDE. Most of the information provided by these resources can also be provided by informational entry points. Whenever possible, the informational entry points should be used. Future versions of the IDE may not support these resources.

For details about informational entry points, see “[Informational Entry Points](#)” on page 65 and “[Compiler and Linker-Specific Entry Points](#)” on page 206.

The resources documented in this chapter describe:

- the features a compiler or linker supports;
- which platforms or processors the plug-in generates software for; the file types normally associated with a plug-in;
- the internal and localized names for a plug-in;
- the family grouping of a plug-in;
- and the plug-in’s executable code.

This chapter describes the following resources:

- [‘cfrg’ Resource and PowerPC Executable Code](#)

- '[Dhlp' Resource](#) – Additional resources which implement the symbol unmangling and browser type name entry points
- '[Drop' Resource](#) – The 68K code implementing a plug-in's main entry point.
- '[EMap' Resource](#) – Specifies the types of files associated with a plug-in by default.
- '[Flag' Resource](#) – Specifies the capabilities of the plug-in to the IDE. Format varies according to plug-in type.
- '[Fmly' Resource](#) –
- '[Targ' Resource](#) –
- '[STR' Resource](#) –
- '[STR#' Resource](#) –

## Resource Descriptions

### 'cfrg' Resource and PowerPC Executable Code

Purpose	Describe and contain PowerPC executable code.
Remarks	The executable part of a plug-in that runs on PowerPC is described by a 'cfrg' resource and is contained in a Code Fragment Manager code fragment, usually residing in the plug-in file's data fork. This code fragment exports entry points for handling requests. Optionally, this code fragment may also export entry points for handling symbol unmangling and browser symbol generation.  An appropriate 'cfrg' resource and PowerPC code for an IDE plug-in are generated by the CodeWarrior IDE when making a project that generates a PowerPC shared library.
See also	<a href="#">"Plug-in Binary Formats and Installation" on page 57</a>

### 'Dhlp' Resource

Purpose	Contains 68K executable code for a plug-in's symbol unmangling entry point or a plug-in's custom browser symbol type names entry point.
---------	---

**Remarks** The executable part of a plug-in that provides symbol unmangling or returns browser symbol names in a non-CFM (Code Fragment Manager) 68K plug-in is contained in a 'Dhlp' resource.

There are two kinds of 'Dhlp' resource:

- A 'Dhlp' resource for returning type names of compiler-specific browser symbols. Such resources should be named 'GetCompilerBrSymbols' and should only be provided by compilers.
- A 'Dhlp' resource for symbol unmangling. Such resources should be named 'Helper\_Unmangle' and should only be provided by linkers.

A 'Dhlp' resource is an executable A4-based 68K code resource. To create a 'Dhlp' code resource, create a project that compiles the plug-in source code to generate a 68K code resource of type 'Dhlp'. Add the resulting code resource to the PowerPC compiler plug-in project.

**See also** ["Plug-in Binary Formats and Installation" on page 57](#)

["'Drop' Resource" on page 643](#)

["'Compiler Browser Symbol Entry Point" on page 209](#)

["'Linker Symbol Unmangling Entry Point" on page 210](#)

["'Helper\\_GetCompilerBrSymbols Entry Point" on page 270](#)

["'Helper\\_Unmangle Entry Point" on page 272](#)

## **'Drop' Resource**

**Purpose** Contains 68K executable code for the plug-in's main entry point.

**Remarks** The executable part of a plug-in that runs on 68K is contained in a 'Drop' resource. A 'Drop' resource is an executable A4-based 68K code resource that contains the plug-in executable code.

To create a 'Drop' 68K code resource, create a project that compiles the plug-in source code to generate a 68K code resource of type 'Drop'. Add the resulting code resource to the PowerPC compiler plug-in project.

The resource name should match the plug-in file name. The IDE loads the code resource by name; its resource ID is not important.

- See also
- [“Plug-in Binary Formats and Installation” on page 57](#)
  - [“Dhlp’ Resource” on page 642](#)
  - [“Compiler Browser Symbol Entry Point” on page 209](#)
  - [“Linker Symbol Unmangling Entry Point” on page 210](#)
  - [“Helper GetCompilerBrSymbols Entry Point” on page 270](#)
  - [“Helper Unmangle Entry Point” on page 272](#)

## **'EMap' Resource**

Purpose	Specifies default file mappings for a plug-in.
Definition	<pre>#include &lt;CWPlugins.r&gt;  type 'EMap' {     integer    kCurrentResourceVersion = 1;      integer = \$\$CountOf(Mappings);     array Mappings {         literal longint    none     = 0,                            text      = 'TEXT';         pstring[31] none = "";         boolean      dontPrecompile, doPrecompile;         boolean      notLaunchable, isLaunchable;         boolean      notResourceFile, isResourceFile;         boolean      handledByMake, ignoredByMake;         // reserved fields     }; };  }</pre>
Remarks	A 'EMap' resource contains an array of default entries that describe the kinds of files a compiler plug-in processes. Each element in the array contains a field for a file's type, its file name extension, and boolean flags specifying if the file is precompilable, is a resource file, can be opened in the project window, and is ignored by a <b>Make</b> operation.

In each item of the array, specify a file's type as a four-byte value and its file name extension as a Pascal string.

Use `doPrecompile` if the compiler knows how to precompile files of this type.

Use `isLaunchable` if the IDE should open the file using the Finder when the user double-clicks files of this type in the project window. A file of type 'TEXT' is opened in an IDE editor window if it is specified as not launchable and is opened by its creator if it is specified as launchable.

Use `isResourceFile` if the file contains binary resources that should be copied into the target executable. Do not use `isResourceFile` for uncompiled, textual resource files.

Use `handledByMake` if the IDE should ignore files of this type during a **Make** operation. Changes to files ignored during make will not affect a project's built status.

Files which are not processed by a compiler, but which are related to it, should not be specified in a compiler's 'EMap' resource. Instead, place these entries, such as those specifying documentation or object and resource data files, in the 'EMap' resource of the linker that processes the compiler's output.

If the file mapping settings for a target in a project do not have an entry for a plug-in, the IDE adds the mappings listed in the plug-in's 'EMap' resource to the target's file mapping settings. If an entry for the plug-in already appears in the target's file mapping settings, the compiler's 'EMap' resource is ignored completely. File mapping settings are never replaced by the contents of an 'EMap' resource.

The resource name should match the plug-in file name. The IDE loads the resource by name; its resource ID is not important.

See also ["Specifying File Mappings" on page 206](#)

## 'Flag' Resource

Purpose Specifies information about the plug-in's capabilities.

Definition `#include <CWPlugins.r>`

## Mac OS Plug-in Resource Formats

### 'Flag' Resource

---

```
type 'Flag' {
    // resource version
    integer    kCurrentResourceVersion = 3;

    switch
    {
        case Compiler:
            key literal    longint = 'Comp';
            switch
            {
                case VERSION10API:
                    key integer = 10;
                    boolean   doesntGenerateCode,
                               generatesCode;
                    boolean   doesntGenerateResources,
                               generatesResources;
                    boolean   cantPreprocess,
                               canPreprocess;
                    boolean   cantPrecompile,
                               canPrecompile;
                    boolean   isntPascal,
                               isPascal;
                    boolean   cantImport,
                               canImport;
                    boolean   cantDisassemble,
                               canDisassemble;
                    boolean   isntPersistent,
                               isPersistent;
                    boolean   dontAllowDuplicateFileNames,
                               allowDuplicateFileNames;
                    boolean   isntMultipleTargetAware,
                               isMultipleTargetAware;
                    boolean   isntMultiprocessingAware,
                               isMultiprocessingAware;
                    boolean   doesntUseTargetStorage,
                               usesTargetStorage;
                    boolean   doesntHaveCompSpecificBrSymbols,
                               hasCompSpecificBrSymbols;
                    boolean   dontAlwaysReload,
                               alwaysReload;
                    boolean   doesntWantBuildStartedRequest,
                               wantsBuildStartedRequest;
            }
    }
}
```

```
boolean
    doesntWantTargetBuildStartedRequest,
    wantsTargetBuildStartedRequest;
boolean
    doesntWantSubprojectBuildStartedRequest,
    wantsSubprojectBuildStartedRequest;
boolean
    doesntWantFileListBuildStartedRequest,
    wantsFileListBuildStartedRequest;

// reserved fields

// language type
literal longint
    CPPLanguage      = 'c++ ',
    PascalLanguage   = 'pasc',
    RezLanguage      = 'rez ',
    JavaLanguage     = 'java',
    UnknownLanguage  = '????';

integer newestAPIVersion;

};

case Linker:
    key literal longint = 'Link';
    switch
    {
        case VERSION10API:
            key integer = 10;
            boolean    canDisassemble,
                        cantDisassemble;
            boolean    isntPostLinker,
                        isPostLinker;
            boolean    dontAllowDuplicateFileNames,
                        allowDuplicateFileNames;
            boolean    isntMultipleTargetAware,
                        isMultipleTargetAware;
            boolean    isntPreLinker,
                        isPreLinker;
            boolean    doesntUseTargetStorage,
                        usesTargetStorage;
```

## Mac OS Plug-in Resource Formats

## *'Flag' Resource*

```
    boolean      doesntUnmangleNames,
                 unmanglesNames;
    boolean      dontAlwaysReload,
                 alwaysReload;
    boolean      doesntWantBuildStartedRequest,
                 wantsBuildStartedRequest;
    boolean
        doesntWantTargetBuildStartedRequest,
        wantsTargetBuildStartedRequest;
    boolean
        doesntWantSubprojectBuildStartedRequest,
        wantsSubprojectBuildStartedRequest;
    boolean      reserved;      // 12
    boolean
        doesntWantTargetLinkStartedRequest,
        wantsTargetLinkStartedRequest;
    boolean      doesntWantPreRunRequest,
                 wantsPreRunRequest;

    // reserved fields

    longint language = 0;
    integer newestAPIVersion;
};

};
```

**Remarks** The fields in this resource specify:

- The version of the 'Flag' resource.
  - The type of the plug-in: compiler, linker, or panel.
  - The API version the plug-in supports.

For compilers, a 'Flag' contains these additional fields:

- Whether or not the compiler generates object data.
  - Whether or not the compiler generates resource data.
  - Whether or not the compiler preprocesses source code.
  - Whether or not the compiler precompiles source code.
  - Whether or not the compiler can import libraries.

Use `canImport` for compilers that translate object code from one format to another.

- Whether or not the compiler disassembles object code into human-readable text.
- Whether or not the compiler should stay resident in memory when the IDE is finished using it.
- Whether or not the compiler supports files in the same target that have the same name
- Whether or not the compiler can be used with more than one target.

When this flag is true, the plug-in does not store any target-related information in global data. If this flag is true, the plug-in will be unloaded and re-loaded between each call. The preferred way to store target-related data is described in ["Managing Target Storage" on page 230](#).

- Whether or not the compiler can be run in a multiprocessing thread.
- Whether or not the compiler uses per-target storage.
- Whether or not the compiler generates compiler-specific browser symbols

For linkers and post linkers, a 'Flag' contains these additional fields:

- Whether or not the linker disassembles object code into text.
- Whether or not the linker is a pre-linker, regular linker, or a post-linker.
- Whether or not the linker unmangles symbols.

The resource name should match the plug-in file name. The IDE loads the flag resource by name; its resource ID is not important.

---

**WARNING!**

At the time of this writing, Rez source was not available for the version 3 resource format described by the Resorcerer template provided in the SDK. The information here applies to version 2 resources, only.

---

See also

["Specifying Plug-in Capabilities" on page 65](#)

## 'Fmly' Resource

Purpose      Specifies a new group of settings panels.

Definition    #include <CWPlugins.r>

```
type 'Fmly' {
    // resource version
    integer    kCurrentResourceVersion = 1;

    // family ID
    literal longint Project          = 'proj',
              FrontEnd        = 'fend',
              BackEnd         = 'bend',
              Browser         = 'brow',
              Editor          = 'edit',
              Debugger        = 'dbug',
              Linker          = 'link',
              Version Control = 'vcs ',
              Miscellaneous   = '*****';

    // family name
    pstring[63];
};
```

Remarks     If a compiler or linker plug-in has settings panels that can't be logically grouped with the default settings panel families, use a 'Fmly' resource that specifies

- a new 4-byte family ID
- the name of the family that appears in the settings panel

The IDE places settings panels with this ID in a new group specified by the 'Fmly' resource.

The resource name should match the plug-in file name. The IDE loads the family resource by name; its resource ID is not important.

See also    ["Specifying Associated Settings Panels" on page 72](#)

## 'Targ' Resource

Purpose     Specifies the processors and operating systems that a plug-in generates software for.

Definition

```
#include <CWPlugins.r>

type 'Targ' {
    // resource version
    integer kCurrentResourceVersion = 1;

    // supported CPUs
    integer = $$CountOf(CPUs);
    array CPUs {
        literal longint PowerPC = 'ppc',
        MC680x0 = '68k',
        i80x86 = '8086',
        MIPS = 'mips',
        V800 = 'v800',
        V850 = 'v850',
        mCore = 'Core',
        SH = 'SH',
        Java = 'java',
        Any = '*****';
    };

    // supported operating systems
    integer = $$CountOf(OperatingSystems);
    array OperatingSystems {
        literal longint MacOS = 'mac',
        Windows = 'wint',
        MagicCap = 'mcap',
        MIPS = 'mips',
        Be = 'be',
        Any = '*****';
    };
};
```

Remarks

A 'Targ' resource lists the processors and operating systems supported by a plug-in. The first list in the 'Targ' array contains one or more entries for the processors the compiler generates data for. The second list contains one or more entries for the operating systems the compiler generates data for.

The wildcard value, Any, may be used to indicate that a plug-in generates data for any processor or any operating system.

The resource name should match the plug-in file name. The IDE loads the target resource by name; its resource ID is not important.

See also [“Specifying Target Platforms” on page 208](#)

## 'STR ' Resource

Purpose Specifies the localized name of a plug-in.

Definition

```
#include <Types.r>

resource 'STR' {
    pstring;
}
```

Remarks A 'STR' resource contains the localized name of the plug-in that appears in the interface of the IDE (for example, in the **Target Settings** dialog).

The name of this resource should match the plug-in's file name with “Name” (note initial space) appended to it. The IDE loads the string by name; the resource ID is not important.

For example, the French localized version of a PowerPC Lisp compiler named “Lisp PowerPC” would have a 'STR' resource that contains the text “Lisp pour PowerPC”. The name of this resource would be “Lisp PowerPC Name”.

See also [“Specifying a Plug-in's Dropin and Display Names” on page 70](#)

## 'STR#' Resource

Purpose Specifies the settings panels associated with a compiler or linker.

Definition

```
#include <Types.r>

type 'STR#' {
    integer = $$Countof(StringArray);
    array StringArray {
        pstring;
    };
}
```

Remarks A 'STR#' resource lists the settings panels a plug-in uses by name.

The name of the resource should match the plug-in's file name with "Panels" (note initial space) appended to it. The IDE loads the string by name; the resource ID is not important.

A PowerPC Lisp compiler would have a 'STR#' resource named "Lisp PowerPC Panels". Its contents would list the preference panels it needs information from, such as: "Lisp Syntax", "Lisp Code Generation", "Lisp Warnings".

See also ["Specifying Associated Settings Panels" on page 72](#)

## **Mac OS Plug-in Resource Formats**

'STR#' Resource

---

# Mac OS CodeWarrior Scripting

---

This chapter introduces and discusses the Apple Event and scripting support provided by the CodeWarrior IDE.

## CodeWarrior Mac OS Scripting Overview

This chapter discusses the scripting and Apple Event commands and classes supported in CodeWarrior. You should read this chapter if you would like to enhance and extend the capabilities of the CodeWarrior IDE to do new things.

The CodeWarrior IDE supports Apple Events. By scripting these Apple Events using AppleScript or another scripting editor, such as Frontier, it is possible to execute many CodeWarrior IDE commands without using the IDE directly. Scripting the CodeWarrior IDE is a way to automate repetitive tasks that do not need user interaction. There are many exciting things that you can do with AppleScript to harness the power of the IDE, such as automate builds, generate files automatically, and configure settings.

If you are primarily interested in writing scripts that manipulate and automate the IDE, then you are probably most interested in using AppleScript to put together an ensemble of Apple Events. If you would like to write program code to drive the CodeWarrior IDE from within your own computer program or tools, then you are probably most interested in the lower-levels of Apple Events, and not in AppleScript. This chapter is oriented toward working with AppleScript, but there is a discussion of low-level Apple Event coding in [“Coding with CodeWarrior IDE and Apple Events” on page 729.](#)

**TIP** Look at the AppleScripts in the (Scripts) folder of the Metrowerks CodeWarrior folder for lots of cool AppleScripts. Reviewing these scripts will save you time when learning to write your own.

---

This chapter is not a tutorial. If you want to learn how to edit, save, and run AppleScripts, you will not find the information here. Instead, refer to other tools and sources of information listed in “[AppleScript Tools and Reference Material](#)” on page 656 for more information.

The topics in this chapter are:

- [AppleScript Tools and Reference Material](#)
- [Writing Your First CodeWarrior IDE AppleScript](#)
- [CodeWarrior IDE AppleScript Events](#)
- [CodeWarrior IDE AppleScript Classes](#)
- [Coding with CodeWarrior IDE and Apple Events](#)

**TIP** You can run AppleScripts from within the CodeWarrior IDE. To learn about how to do this, read the section of the *IDE User Guide* manual that discusses this topic.

---

## AppleScript Tools and Reference Material

You can find the tools provided by Apple Computer for editing and running AppleScripts on the web. On the internet, Apple Computer maintains a web site for AppleScript issues, as well as email lists of AppleScript topics. Point your web browser at:

<http://www.apple.com/applescript>

You will need a Script Editor, such as Apple’s Script Editor that comes with the AppleScript software on the CodeWarrior Reference CD. Script Editor is preinstalled on most later-model Mac OS computers.

Other editing and debugging tools are available from third-party vendors. These products are worth evaluating if you are going to do much AppleScripting. You can find a listing at:

<http://www.scriptweb.com>

If you are a subscriber to the Apple Developer CD program or Apple Developer Mailing, you will find good information on Apple Events and AppleScripting on the CDs too.

For more information on using and writing AppleScripts, you may want to consult other publications, such as:

- *AppleScript Language Guide: English Dialect* (Addison-Wesley)
- *Danny Goodman's AppleScript Handbook* (Random House)
- *The Tao of AppleScript* (Hayden Books)
- *Applied Mac Scripting* (M & T Books)

For information on more advanced topics such as writing your own Scripting Additions, or how to use the standard Scripting Additions, refer to *AppleScript Scripting Additions Guide* (Apple Computer).

Finally, for documentation on using low-level AppleEvents in program code, refer to *Inside Macintosh: Interapplication Communication* (published by Addison-Wesley).

## Writing Your First CodeWarrior IDE AppleScript

To get started with AppleScript and the CodeWarrior IDE, let's take a look at a simple script that opens the IDE, brings it to the foreground on the Mac, opens a project, removes the binaries, and starts a build of the project. This script, shown in [Listing A.1](#), is something that could be double-clicked to automatically do all these operations unattended.

---

**TIP** You might want to rename your CodeWarrior IDE application to just CodeWarrior IDE instead of CodeWarrior IDE 4.0.4. This makes it easier to migrate your scripts as versions of the IDE change.

---

### **Listing 18.1 My First CodeWarrior AppleScript**

---

```
tell application "CodeWarrior IDE" (* go! *)
activate (* bring CW to the front *)
open file "SD:MyProj:MyProject.mcp"
Remove Binaries
Make Project
end tell
```

---

You can imagine how convenient it will be to automate many tasks with AppleScript from this short example. Try entering this script in your Editor and get it to run. After getting this short example to run, you will probably be motivated to try some more extensive examples.

---

**TIP** If you are using Apple's Script Editor instead of a third-party script editor, you might want to increase the size of the memory partition for Script Editor while in the Finder. This is because Apple ships the application with a very small default memory partition. If you notice strange behavior while running your scripts, you might try increasing the memory for Script Editor to see if the problems go away.

---

## **CodeWarrior IDE AppleScript Events**

In general, Apple Events are grouped in categories or “suites” of events that provide some common theme for the events. There is a “Required” suite of events that includes open, print, quit and run. All scriptable applications should support the required suite. There are other suites of events defined in the *Apple Event Registry* document. In addition, there are other suites of events that are application-specific.

For many of the things that you probably want to do with the CodeWarrior IDE, it really isn't a concern which suite an event is from most of the time. However, you can view the “dictionary” of Apple Events that an application supports using the Open Dictionary command of your Script Editor. See the documentation that came with your editor for information about viewing the dictionary.

In this section, we discuss how to handle errors in AppleScript, and several categories of events that you can use to control the CodeWarrior IDE.

- [Processing Errors](#)
- [Required Events](#)
- [File Handling Events](#)
- [Building Events](#)
- [Status/Query Events](#)
- [Navigation Events](#)

### **Parameters**

Some Apple Events listed in this section require a parameter called *filename-list*. The *filename-list* represents a single filename or a list of filenames and/or aliases. A single filename is a quoted character string. A list of filenames is enclosed in braces, {}, with the filenames separated by commas.

---

#### **Listing 18.2    Example values for *filename***

---

```
"myprogram.c"  
{ "startup.p", "printout.p", "drawbox.p" }  
{ "codechecker.cpp" }  
{ "HD:CodeWarrior f:My Projects:hello.c" }  
file "myprogram.c"  
alias "myprogram.c"
```

---

### **Processing Errors**

When an AppleEvent is sent to CodeWarrior, errors may be returned to the script. Errors are not always evil occurrences, as sometimes you will want to trap errors to make your script do other things in response to the current conditions. For example, you can trap a file-not-found error and direct the CodeWarrior IDE to perform an alternate action as a result. Errors can be generated from the operating system, or from the application you're trying to script. Errors for the operating system are documented in Appendix C of the *AppleScript Language Guide*. Errors generated by the CodeWarrior IDE are documented here.

Errors are usually returned through the normal Error-return channel. However, for events that process a list of files, the errors are returned in the result (a built-in AppleScript variable). The list, with each member corresponding to an input file, is returned as the event's result with each list member.

In addition to operating system errors, such as out of memory errors, the error codes listed in [Table 18.1](#) may also be returned.

**Table 18.1** **CodeWarrior keyAEResult result codes (typeShortInteger)**

Name	Value
noErr	0
errShell_ActionFailed	1
errShell_FileNotFound	2
errShell_DuplicateFile	3
errShell_CompilerError	4
errShell_MakeFailed (compile or link error)	5
errShell_NoOpenProject	6
errShell_WindowNotOpen	7
errShell_SegmentNotFound	8
errShell_TargetNotFound	9
errShell_ProjectNotFound	10
errShell_DisabledInLightIDE	11

These error codes are self-evident, with the exception of errShell\_DisabledInLightIDE which occurs when trying to invoke scripting on a version of the IDE software that does not support scripting operations. This could occur with evaluation, free copies, or other restricted releases of the CodeWarrior IDE.

The result parameter, keyAEResult, is not set if there is an error while interpreting the AppleEvent (running out of memory).

supplying a bad parameter type, and so on). In such cases, an error code is returned in the standard `keyErrorNumber` parameter.

[Listing 18.3](#) gives an example of an AppleScript that handles an error.

### **Listing 18.3 Error handling in AppleScript**

---

```
try
    tell application "CodeWarrior IDE"
        set doclist to (Get Open Documents)
    end tell
on error number errnum
    display dialog "Bummer!" & errnum
end try
```

---

To see more examples of error handling in scripts, review some of the scripts in the (Scripts) folder, in the same folder as your CodeWarrior IDE application.

## **Required Events**

There are four events that are required for every application that claims to be AppleScriptable. This section discusses these four events and their syntax.

The events covered in this section are:

- [Open](#)
- [Print](#)
- [Quit](#)
- [Run](#)

### **Open**

Purpose    This event tells the CodeWarrior IDE to open the specified files.

---

Open *filename-list* [ converting ] *expression*

---

If `converting` is specified, any project files that were created with previous versions of the CodeWarrior IDE will be updated.

**Listing 18.4 Example for Open**

---

```
Open "HD:MyProject.mcp" converting yes
Open "HD:MyProject.mcp"
```

---

**Print**

Purpose This event tells the CodeWarrior IDE to print the specified files.

---

**Print *filename-list***

---

**Quit**

Purpose This event tells the CodeWarrior IDE to quit.

**Run**

Purpose This event is sent to an application when it is double-clicked. Upon receiving the event, the application should launch itself.

## File Handling Events

You will want to use the File Handling Apple Events of the CodeWarrior IDE to do things like add and remove files in a project, close a window, create and close a project, and save copies of files.

Here are the events covered in this section:

- [Add Files](#)
- [Close Project](#)
- [Close Window](#)
- [Create Project](#)
- [Remove Files](#)
- [Save Error Window As](#)
- [Select](#)
- [Set Modification Date](#)

### Add Files

Purpose Adds the specified files to the current project.

---

---

Add Files *filename-list* [ to segment *number* ]

---

Description	This event is equivalent to the <b>Add Files</b> command in the <b>Project Menu</b> . The <i>filename-list</i> parameter describes a single filename or list of filenames to add the current project. The optional <i>to segment</i> parameter specifies the segment in the project in which to add the files. Replace <i>number</i> with the segment number to place the files in. The default is to create a new segment.
Returns	A list of errors. The result code for each file added to the project can either return the value of an OSErr (Operating System Error) or one of the following values: <ul style="list-style-type: none"><li>• noerr</li><li>• errShell_FileNotFound</li><li>• errShell_DuplicateFile</li><li>• errShell_NoOpenProject</li></ul>

### **Listing 18.5 Examples for Add Files**

---

```
Add Files "MyFile.c"
Add Files "MyFile.c" to segment 2
Add Files {"MyFile.c", "MyFile2.c"}
Add Files {"MyFile.c", "MyFile2.c"} to segment 3
```

---

### **Close Project**

Purpose Closes the current project.

---

Close Project

---

Returns None.

### **Close Window**

Purpose Closes editor windows.

---

Close Window *filename* [ saving *status* ]

---

Description	This event is equivalent to the <b>Close</b> command in the <b>File Menu</b> . If <i>filename</i> is a string, Close Window first tries to find the first
-------------	---

matching window name, searching front to back. If no window name matches *filename*, then Close Window causes a search for a matching filename.

To specify read/only windows, add “ [r/o 1]” to the end of *filename*.

---

```
Close Window "hello.c [r/o 1]"
```

---

[Table 18.2](#) lists file saving options used with the close window AppleScript command.

**Table 18.2 Saving options ('savo')**

Name	Property Code
yes (Save changes)	'yes '
no (Do not save changes)	'no '
ask (Ask the user whether to save)	'ask '

The optional *saving* parameter determines if the windows contents are saved before closing the window.

- If *status* is yes, save the window's contents.
- If *status* is no, discard changes made to the file.
- If *status* is ask, prompt the user whether or not to save the file.

Returns None.

### **Listing 18.6 Example for Close Window**

---

```
Close Window "untitled"
-- Closes first "untitled" window.
Close Window "hello.c" saving yes
Close Window "main.c [r/o 1]"
-- Closes read/only window.
```

---

### **Create Project**

Purpose Creates a new project file.

---

Create Project *filename* [ from stationery ]*alias*

---

Description     Performs the **New Project** command in the **File Menu**. Replace *filename* with a single filename for the new project. If *from stationery* is specified, project stationery specified by *alias* is used to create the new project.

Returns     The result code can have the following values:

- noErr
- errShell\_ActionFailed

### **Listing 18.7 Examples for Create Project**

---

```
Create Project "HardDisk:Projects:MyProject.pu"
Create Project "Foobe" from ¬
stationery "HD:Dev:Metrowerks" & ¬
"CodeWarrior:(Project Stationery):MacOS:C/C++:" & ¬
"Basic Toolbox:Basic Toolbox.mcp"
Create Project "::sample project:sample.mcp"
```

---

## **Remove Files**

Purpose     Removes the specified file(s) from the current project.

---

Remove Files *filename*

---

Description     Performs the equivalent of the **Remove Selected Items** command in the **Project Menu**. Replace *filename* with a single file or a list of files to remove from the current project.

Returns     A list of errors. The result code can have one of the following values:

- noErr
- errShell\_FileNotFound
- errShell\_NoOpenProject

### **Listing 18.8 Examples for Remove Files**

---

```
Remove Files "MyFile.c"
Remove Files { "MyFile.c", "YourFile.c" }
```

---

### **Save Error Window As**

Purpose     Saves the contents of the message window as a text file.

---

Save Error Window as *filename*

---

Description     Performs the equivalent of the **Save A Copy As** command in the **File Menu** when the Message window is active. The contents of the Message window are saved with the name *filename*.

Returns     None.

### **Listing 18.9   Example for Save Error Window As**

---

```
Precompile "main.pch"  
Save Error Window As "main.pch results"
```

---

### **Select**

Purpose     Selects an object from an open document in the CodeWarrior Editor.

---

Select *reference*

---

Description     The parameter *reference* is the object to select.

### **Listing 18.10   Example for Select**

---

```
Select text from character 5 to character 10 ¬  
of line 8 of document 1
```

---

### **Set Modification Date**

Purpose     Sets the modification date of the specified file(s).

---

Set Modification Date *filename-list* to *date*

---

Returns     A list of results of type short integer, or, if the **ExternalEditor** option is specified, a list of records of type '**ErrM**'.

## **Building Events**

Here are the events discussed in this section:

---

- [Build](#)
- [Check](#)
- [Check Syntax](#)
- [Compile](#)
- [Compile File](#)
- [Disassemble File](#)
- [Make Project](#)
- [Precompile](#)
- [Preprocess](#)
- [Remove Binaries](#)
- [Remove Object Code](#)
- [Run](#)
- [Run Project](#)
- [Touch](#)
- [Update Project](#)

## **Build**

Purpose     Builds the current target or project.

---

### **Build**

Description     Performs the **Make** command in the **Project Menu**.

Returns     By default, **Build** returns nothing.

## **Check**

Purpose     Checks the syntax of the specified file(s).

---

### **Check***filename-list*

Description     This event is equivalent to performing the **Check Syntax** command in the **Project Menu**. Replace *filename-list* with a single filename or a list of filenames in the project.

---

By default, Check returns a list of short integer result codes for each file checked. A result code can either be the value of an OSerr (Operating System Error) or one of the following values:

- noerr
- errShell\_FileNotFound
- errShell\_CompilerError
- errShell\_NoOpenProject

Returns A list of results of type short integer.

### **Listing 18.11 Examples for Check File Syntax**

---

```
Check File Syntax "MyFile.c"  
Check File Syntax {"MyFile.c", "YourFile.c"} with ExternalEditor
```

---

### **Check Syntax**

Purpose Checks the syntax of the specified file(s).

---

```
Check Syntax filename-list [with ExternalEditor]
```

---

Description This event is equivalent to performing the **Check Syntax** command in the **Project Menu**. Replace *filename-list* with a single filename or a list of filenames in the project.

By default, Check Syntax returns a list of short integer result codes for each file checked. A result code can either be the value of an OSerr (Operating System Error) or one of the following values:

- noerr
- errShell\_FileNotFound
- errShell\_CompilerError
- errShell\_NoOpenProject

If the ExternalEditor option is used, the environment returns the Message window contents instead of the usual list of short integer results. The AppleEvent keyword for ExternalEditor is 'Errs'. It takes a boolean parameter.

Returns A list of results of type short integer, or, if the ExternalEditor option is specified, a list of records of type 'ErrM'.

---

### **Listing 18.12 Examples for Check Syntax**

---

```
Check Syntax "MyFile.c"  
Check Syntax {"MyFile.c", "YourFile.c"} with ExternalEditor
```

---

### **Compile**

Purpose    Compiles the specified file(s).

---

```
Compile filename-list [with ExternalEditor]
```

---

Description    This event is equivalent to performing the **Compile** command on the **Project Menu**. Replace *filename-list* with a single filename or a list of filenames.

By default, **Compile** returns a list of short integer result codes for each compiled file. A result code can be an OSerr value (Operating System Error) or one of the following:

- noerr
- errShell\_FileNotFound
- errShell\_CompilerError
- errShell\_NoOpenProject

If the **ExternalEditor** option is specified, the environment returns the Message window contents as a list of '**ErrM**' objects.

Returns    A list of errors of type short integer or, if **ExternalEditor** is specified, of type '**ErrM**'.

### **Listing 18.13 Examples for Compile**

---

```
Compile "MyFile.c"  
Compile {"MyFile.c", "YourFile.c"}
```

---

### **Compile File**

Purpose    Compiles the specified file(s).

---

```
Compile File filename-list
```

---

Description	This event is equivalent to performing the <b>Compile</b> command on the <b>Project Menu</b> . Replace <i>filename-list</i> with a single filename or a list of filenames.  By default, <b>Compile</b> returns a list of short integer result codes for each compiled file. A result code can be an OSerr value (Operating System Error) or one of the following:
	<ul style="list-style-type: none"><li>• noerr</li><li>• errShell_FileNotFound</li><li>• errShell_CompilerError</li><li>• errShell_NoOpenProject</li></ul>

Returns A list of errors of type short integer.

#### **Listing 18.14 Examples for Compile**

---

```
Compile File "MyFile.c"  
Compile File {"MyFile.c", "YourFile.c"}
```

---

### **Disassemble File**

Purpose Disassembles the specified file(s).

---

Disassemble File *filename-list*

---

Description	This event is equivalent to performing the <b>Disassemble</b> command on the <b>Project Menu</b> . Replace <i>filename-list</i> with a single filename or a list of filenames.
Returns	A list of errors of type short integer.

#### **Listing 18.15 Examples for Disassemble File**

---

```
Disassemble File "MyFile.c"  
Disassemble File {"MyFile.c", "YourFile.c"}
```

---

### **Make Project**

Purpose Makes the current project.

---

Make Project [with ExternalEditor]

---

Description	Performs the <b>Make</b> command in the <b>Project Menu</b> .  If the <code>ExternalEditor</code> option is used, the environment returns the Message window contents.
Returns	By default, <code>Make</code> Project returns nothing. If <code>ExternalEditor</code> is specified, <code>Make</code> Project returns a list of errors of type ' <code>ErrM</code> '.

### **Precompile**

Purpose	Precompiles the specified file.
---------	---------------------------------

---

```
Precompile source saving as destination [ with ExternalEditor ]
```

---

Description	This event is equivalent to the <b>Precompile</b> command in the <b>Project Menu</b> . Replace <i>source</i> with the name of a file to precompile. Replace <i>destination</i> with the filename of the precompiled header.  If the <code>ExternalEditor</code> option is used, the environment returns the Message window contents.
Returns	By default, <code>Precompile</code> returns nothing. If <code>ExternalEditor</code> is specified, <code>Precompile</code> returns a list of errors of type ' <code>ErrM</code> '.

### **Listing 18.16 Example for Precompile**

---

```
Precompile "MyHeaders.pch" saving as "MyHeaders.mch"
Precompile "tip.pch" saving as "tip.mch" with ExternalEditor
```

---

### **Preprocess**

Purpose	Preprocesses the specified file.
---------	----------------------------------

---

```
Preprocess source [ with ExternalEditor ]
```

---

Description	This event is equivalent to the <b>Preprocess</b> command in the <b>Project Menu</b> . Replace <i>source</i> with the name of a file to preprocess.  If the <code>ExternalEditor</code> option is used, the environment returns the Message window contents.
Returns	By default, <code>Preprocess</code> returns nothing. If <code>ExternalEditor</code> is specified, <code>Preprocess</code> returns a list of errors of type ' <code>ErrM</code> '.

### **Listing 18.17 Examples for Preprocess**

---

```
Preprocess "MyHeaders.c"  
Preprocess "tip.c" with ExternalEditor
```

---

### **Remove Binaries**

Purpose    Removes the binary object code from the current project.

---

Remove Binaries

---

Description    Performs the equivalent of the **Remove Object Code** command in the **Project Menu**.

Returns    None.

### **Remove Object Code**

Purpose    Removes the binary object code from the current target or project.

---

Remove Object Code

---

Description    Performs the equivalent of the **Remove Object Code** command in the **Project Menu**.

Returns    None.

### **Run**

Purpose    Runs the current project or target.

---

Run

---

Description    Performs the equivalent of the **Run** command in the **Project Menu**. This event builds then executes the current target if there are no compile or link errors.

Returns    By default, Run Project returns nothing.

### **Run Project**

Purpose    Runs the current project

---

Run Project [ with ExternalEditor ] [ with SourceDebugger ]

---

Description	Performs the equivalent of the <b>Run</b> command in the <b>Project Menu</b> . This event builds then executes the current project if there are no compile or link errors.
	If the <code>ExternalEditor</code> option is used, the environment returns the Message window contents.
Returns	If the <code>SourceDebugger</code> option is used, the environment launches the successfully-built project into the source-level debugger.

#### **Listing 18.18 Examples for Run Project**

---

```
Run Project with SourceDebugger  
Run Project with ExternalEditor
```

---

### **Touch**

Purpose	Touches the specified file(s).
<b>Touch <i>filename</i></b>	

Description	Performs the equivalent of clicking the <b>Touch</b> column in a Project window. Touching a file forces it to be recompiled during a make operation. Replace <i>filename</i> with a single file or a list of files to touch.
	For more on touching a file to be recompiled, consult the <i>IDE User Guide</i> for information on synchronizing files.
Returns	A list of errors. Each result code can have one of the following values: <ul style="list-style-type: none"><li>• <code>noErr</code></li><li>• <code>errShell_FileNotFound</code></li><li>• <code>errShell_NoOpenProject</code></li></ul>

**Listing 18.19 Examples for Touch**

---

```
Touch "MyFile.c"  
Touch { "MyFile.c", "YourFile.c" }
```

---

**Update Project**

Purpose     Updates the current project.

---

```
Update Project [ with ExternalEditor ]
```

---

Description    This command is equivalent to the **Bring Up To Date** command in the **Project Menu**. If the `ExternalEditor` option is used, the environment returns the Message window contents.

Returns      By default, `Update Project` returns nothing. If `ExternalEditor` is specified, `Update Project` returns a list of errors of type '`ErrM`'.

**Status/Query Events**

Here are the events discussed in this section:

- [Get](#)
- [Set](#)
- [Get Definition](#)
- [Get Member Function Names](#)
- [Get Nonsimple Classes](#)
- [Get Open Documents](#)
- [Get Preferences](#)
- [Set Preferences](#)
- [Get Project File](#)
- [Set Project File](#)
- [Set Current Target](#)
- [Set Default Project](#)
- [Get Project Specifier](#)
- [Get Segments](#)
- [Set Segment](#)

- [Is In Project](#)
- [Reset File Paths](#)
- [Close](#)
- [Count](#)
- [Make](#)

### **Get**

Purpose Gets the object referenced.

---

*Get reference [ as list of typeclass ]*

---

Description The parameter *reference* is the object whose data is to be returned. The parameter *typeclass* is the desired types for the data, in order of preference.

### **Set**

Purpose Sets the object referenced.

---

*Set reference to anything*

---

Description The parameter *reference* is the object whose data is to be changed. The parameter *anything* is the new value for the object.

### **Listing 18.20 Example for Set**

---

```
tell application "CodeWarrior IDE" to  
set numClasses to the count of classes
```

---

### **Get Definition**

Purpose Queries the location(s) of a globally-scoped function or data object for the current project.

---

*Get Definition string*

---

Description The *string* is the name of the symbol you are interested in.

Returns Record containing a list of the function information.

---

**Listing 18.21 Example for Get Definition**

---

Get Definition "main"

---

**Get Member Function Names**

Purpose Gets a list of all the member functions of a class object.

---

Get Member Function Names *reference*

---

Returns List containing the information.

**Listing 18.22 Example for Get Member Function Names**

---

Get Member Function Names class "CPowerTelnetApp"

---

**Get Nonsimple Classes**

Purpose Gets a list of all the member functions of a class object.

---

Get Nonsimple Classes

---

Returns List containing the information.

**Listing 18.23 Example for Get Nonsimple Classes**

---

Get Nonsimple Classes class "CPowerTelnetApp"

---

**Get Open Documents**

Purpose Gets the list of open documents.

---

Get Open Documents

---

Returns List of documents in records of type 'docu'. See [Table 18.41](#) for more information

**Get Preferences**

Purpose Gets settings from a panel.

---

Get Preferences [ of *pref-list* ] from panel *panel-name*

---

- Description     The *panel-name* must be the name of the preference panel file and not the name that appears in the preferences window. For example, to set C/C++ Language options, use "C/C++ Compiler" as the *panel-name* and not "C/C++ Language".
- Returns     Record containing a list of the requested preferences. If you do not include *pref-list*, it returns all the preferences for *panel-name*.

#### **Listing 18.24 Examples for Get Preferences**

---

Get Preferences from panel "C/C++ Compiler"  
Get Preferences of {File Name, SIZE Flags} ↵  
from panel "PPC Project"

---

#### **Set Preferences**

Purpose     Specifies the settings for a panel.

---

Set Preferences of panel *panel-name* to *record*

---

- Description     Performs the equivalent of setting options using either the Preferences or Settings windows. This event lets you set the properties of the current project. It is not necessary to specify every preference, those not mentioned in the record retain their settings. The properties for different panels are listed in various tables from this chapter.
- The *panel-name* must be the name of the preference panel file and not the name that appears in the preferences window. For example, to set C/C++ Language options, set *panel-name* to "C/C++ Compiler" and not "C/C++ Language".
- Returns     None.

#### **Listing 18.25 Examples for Set Preferences**

---

Set Preferences of panel "PPC Project" to { ↵  
File Name:"MyProgram", File Creator:"Mine", SIZE Flags:23008 ↵  
}

---

## Mac OS CodeWarrior Scripting

### Status/Query Events

---

```
Set Preferences of panel "C/C++ Compiler" to {  
    Prefix File: "MacHeaders",  
    Activate CPlusPlus: TRUE,  
    Require Function Prototypes: FALSE  
}  
Set Preferences of panel "C/C++ Warnings" to {  
    Extended Error Checking: TRUE  
}
```

---

## Get Project File

Purpose Gets information on a project entry.

---

Get Project File *file-number* segment *seg-number*

---

Returns The information of the specified entry in the current project as a record of type 'SrcF'. The *file-number* parameter specifies a file within its segment or group. The *seg-number* parameter specifies a segment or group within the project. Numbering for both parameters are short integers beginning at 1.

### **Listing 18.26 Examples for Get Project File**

```
get project file 1 segment 1  
    -- First entry in project  
get project file 1 segment 2  
    -- First entry in 2nd segment
```

---

## Set Project File

Purpose Sets information on a project entry.

---

Set Project File *filename* to *record*

---

Description Changes the settings for the specified entry in the open project. The *record* parameter is of type 'SrcF'. Only the symbols and weak link fields are allowed. [Listing A.28](#) shows how to set weak linking for a library InterfaceLib that is in a project.

Returns No returns.

**Listing 18.27 Example for Set Project File**

---

```
Set Project File "InterfaceLib" to {weak link: true}
```

---

**Set Current Target**

Purpose Sets the current target for a project.

---

```
Set Current Target name-of-target
```

---

Description This AppleEvent causes the build target to change. This event would be useful when changing the build target in a project, so that a new target can be built or otherwise operated on with other AppleEvents. To learn more information about setting the current build target, refer to the *IDE User Guide*.

Returns None.

**Listing 18.28 Example for Set Current Target**

---

```
Set Current Target "Muscle Debug"
```

---

**Set Default Project**

Purpose Sets the default project.

---

```
Set Default Project name-of-target
```

---

Description This AppleEvent causes the default project to change. To learn more information about setting a default project, refer to the *IDE User Guide*.

Returns None.

**Listing 18.29 Example for Set Default Project**

---

```
Set Current Project "Muscle.mcp"
```

---

**Get Project Specifier**

Purpose Gets the filename of the project.

---

## Mac OS CodeWarrior Scripting

### Status/Query Events

---

#### Get Project Specifier

---

Returns     The name of the current project.

#### Get Segments

Purpose    Gets the descriptions of all segments/groups in the open project.

---

#### Get Segments

---

Returns    List of documents in records of type 'Seg'. Refer to "[Segment" on page 692](#) for more information

#### Set Segment

Purpose    Sets preferences for the current project.

---

#### Set Segment *number* to *record*

---

Description    Sets information for a segment or group in the open project. Segment numbering starts at 1. The *record* parameter is an object of type 'Seg'. [Listing A.31](#) shows how to rename a segment/group in a project. Refer to "["Segment" on page 692](#) for more information.

Returns    None.

#### **Listing 18.30   Example for Set Segment**

---

Set Segment 1 to {name:"New Sources"}

---

#### Is In Project

Purpose    Are the specified file(s) are in the project?

---

#### Is In Project *filename*

---

Description    Replaces *filename* with a single filename or a list of filenames.

Returns    A list of errors. The result code for each specified file can have the following values:

- noErr if the file is in the project

- `errShell_FileNotFound` if file is not in the project.

### **Listing 18.31 Examples for Is In Project**

---

```
Is In Project "SillyBalls.c"
Is In Project { "SillyBalls.c", "Initialize.c" }
```

---

### **Reset File Paths**

Purpose    Resets access paths for all files belonging to the open project.

---

```
Reset File Paths
```

---

Returns    None.

### **Close**

Purpose    Closes an object. The `saving` and `saving in` parameters are optional, and have the range of values listed here.

---

```
Close reference [ saving yes/no/ask ] [ saving in alias ]
```

---

Returns    None.

### **Count**

Purpose    Counts the number of elements within an object.

---

```
Count reference each type class
```

---

Returns    An integer indicating the number of elements.

### **Make**

Purpose    Makes a new element. The `as` list of, `as`, `with data`, and `with properties` parameters are optional, and have the range of values listed here.

---

```
Make new [ as list of type class ] [ at location reference ] [
with data anything ] [ with properties record ]
```

---

Returns    A reference to the new object(s).

## Navigation Events

Here are the events discussed in this section:

- [Goto Function](#)
- [Goto Line](#)
- [Open Browser](#)

### Goto Function

Purpose Jumps to the specified function defined in the active editor window.

---

Goto Function *name*

---

Description This event is equivalent to selecting a routine name in the active editor window's function pop-up menu. The insertion point does not move when a function is selected with this event.

Returns None.

### Listing 18.32 Examples for Goto Function

---

```
Goto Function "main"  
Goto Function "SkipBlanks"
```

---

### Goto Line

Purpose Jumps to the specified line number in the active editor window.

---

Goto Line *number*

---

Description Goto Line moves the insertion point to the specified line number, *number*, in the active editor window. If the line number specified exceeds the last line number, the insertion point is placed at the last line.

Returns None.

### Listing 18.33 Examples for Goto Line

---

```
Goto Line 1  
Goto Line 493
```

---

## **Open Browser**

**Purpose** Displays a class, member function, or data member object in a single class browser window. You cannot display a procedural function.

---

*Open Browser reference*

---

**Returns** None.

### **Listing 18.34 Example for Open Browser**

---

```
Open Browser class "CPowerTelnetApp"
Open Browser member function 2 of class "CPowerTelnetApp"
```

---

## **CodeWarrior IDE AppleScript Classes**

CodeWarrior events have several classes to let you control the CodeWarrior IDE's actions and option settings. These classes are based on the items available in the **Preferences** and **Target Settings** windows.

The available classes are determined by the compiler (C/C++, Java, or Pascal) and the processor for which it is generating object code (Intel x86 or PowerPC-based Macintosh). In other words, the preference classes available for the CodeWarrior C/C++ compiler that generates object code for the PowerPC are different from those available for the CodeWarrior C compiler that generates x86 object code.

AppleScript classes for the CodeWarrior environment are, for the most part, separated by preference panels, project settings, and then by miscellaneous environment items.

- [Project Classes](#)—contains properties for each aspect of a project, from target parameters to final application settings.
- [Compiler Classes](#)—contains properties for each language and compiler, to configure compilation settings and warnings.
- [CodeGen Classes](#)—contains properties for each possible kind of code that can be generated by the CodeWarrior compilers.
- [Disassembler Classes](#)—contains properties for the disassemblers.

- [Linker Classes](#)—contains properties for the linker settings.
- [Build Classes](#)—contains properties for build environment settings and errors.
- [Browser Classes](#)—contains properties for the code browser.
- [Editor Classes](#)—contains properties for the CodeWarrior IDE text editor.
- [Object Classes](#)—contains properties that describe objects, including data members, classes and base classes, and member functions.
- [Miscellaneous Classes](#)—contains properties that describe miscellaneous aspects of the CodeWarrior IDE environment.

Many options that use a pop-up menu in the preference panel now use an enumerated type to specify their values, instead of an integer. For example, to set the Code Model, you should now use small, smart, or large, instead of 1, 2, or 3.

---

**WARNING!**

Your script will not work if you use integer to set a property that expects a symbol.

---

## Project Classes

- [PowerPC Project](#)
- [Java Project](#)
- [Win32/x86 Project](#)
- [PPC Mach-O Project](#)
- [Access Paths](#)
- [Path Information](#)
- [Target Settings](#)
- [File Mapping Information](#)
- [Segment](#)
- [Project File](#)

### PowerPC Project

[Table 18.3](#) lists the PPC Project class properties.

**Table 18.3 PPC Project Class**

Name	Property Type
Project Type	<ul style="list-style-type: none"> <li>• standard application</li> <li>• CFM68K application</li> <li>• code resource</li> <li>• library</li> <li>• shared library</li> <li>• MPW Tool</li> <li>• Pilot Application</li> <li>• Pilot Code Resource</li> </ul>
File Name	string
File Creator	string
File Type	string
Minimum Size	integer
Preferred Size	integer
SIZE Flags	small integer (SIZE flag bits must be computed as an integer value)
SYM File	string
Resource Name	string
Display Dialogs	boolean
Merge To File	boolean
Resource Flags	small integer
Resource Type	string
Resource ID	small integer
Stack Size	integer

Name	Property Type
Header Type	<ul style="list-style-type: none"><li>• none</li><li>• native</li></ul>
Flatten Resources	<ul style="list-style-type: none"><li>• boolean</li></ul>

### Java Project

[Table 18.4](#) lists the Java Project class properties.

**Table 18.4 Java Project Class**

Name	Property Type
Main Class	string
Java Project Type	<ul style="list-style-type: none"><li>• java applet</li><li>• java application</li><li>• java library</li></ul>
Arguments	string
Compress	boolean
HTML Helper App	string

### Win32/x86 Project

[Table 18.5](#) lists the x86 Project class properties.

**Table 18.5 x86 Project Class**

Name	Property Type
Project Type	<ul style="list-style-type: none"><li>• standard application</li><li>• shared library</li><li>• library</li></ul>
File Name	string
Min Heap Size	integer
Preferred Heap Size	integer

<b>Name</b>	<b>Property Type</b>
Base Address	integer
Max Stack Size	integer
Min Stack Size	integer
Import Library	string

### **PPC Mach-O Project**

This class controls the project features for Mach-O projects.

**Table 18.6    Mach-O Project Class**

<b>Name</b>	<b>Property Type</b>
Project Type	<ul style="list-style-type: none"> <li>• standard application</li> <li>• CFM68K application</li> <li>• code resource</li> <li>• library</li> <li>• shared library</li> <li>• MPW Tool</li> <li>• Pilot Application</li> <li>• Pilot Code Resource</li> </ul>
File Name	string
File Creator	string
File Type	string
Stack Size	integer
Stack Address	integer
Flatten Resources	boolean
Flatten Resources Folder	file
Flatten Resources Name	string

## Access Paths

[Table 18.7](#) lists the properties for the Access Paths Class.

**Table 18.7 Access Paths Class**

Name	Property Type
User Paths	list (of path information records)
Always Full Search	boolean
Convert Paths	boolean
System Paths	list (of path information records)

## Path Information

A path information record may contain the properties shown in [Table 18.8](#). It must contain at least the name field.

**Table 18.8 Path Information Class**

Name	Property Type
name	string
recursive	boolean
origin	<ul style="list-style-type: none"><li>• absolute</li><li>• project relative</li><li>• shell relative</li><li>• system relative</li></ul>
host flags	<ul style="list-style-type: none"><li>• 1 (for the Mac OS)</li><li>• 2 (for Windows)</li></ul>

If you use a string instead of a path information record, CodeWarrior sets recursive to true and origin to project relative.

To clear all the access path entries listed in the Access Paths preference panel, set the User Paths or System Paths property to an empty list. For example, in AppleScript, this statement removes all entries, including the default entries, {Project *f*} and {Compiler *f*}:

---

```
Set Preferences of panel "Access Paths" to ¬
{User Paths: {}, System Paths: {} }
```

---

To add a default entry back, add an access path record with the name set to ":" and with the origin set to project relative (for {Project *f*}) or shell relative (for {Compiler *f*}). For example, this statement sets User Paths to {Project *f*} and System Paths to {Compiler *f*}:

---

```
Set Preferences of panel "Access Paths" to ¬
{User Paths: {{name: ":" , ¬
    origin: project relative}}, ¬
System Paths: {{name: ":" , ¬
    origin: shell relative}} }
```

---

For more information on the meaning of the properties listed in [Table 18.7](#) and [Table 18.8](#), see [“Access Paths” on page 688](#).

### Target Settings

You set the current target parameters in the Target Settings options panel with the properties shown in [Table 18.9](#).

**Table 18.9    Target Settings Class**

Name	Property Type
Target Name	string
Linker	string
Post Linker	string
Output Directory Path	string

Name	Property Type
Output Directory	• absolute
Origin	• project relative • shell relative • system relative
Pre Linker	string

The Target Name string is the name of the target (you choose this name). The Linker string must be the name of one of the files in the Linkers folder of the CodeWarrior Plugins folder. The Post Linker string must be the name of one of the files in the Post Linkers folder of the CodeWarrior Plugins folder. The Output Directory Path string is a string that points to a location on your hard disk where the output files should be placed after linking. You can make this path absolute, or you can make it relative to the location of the project (project relative), compiler (compiler relative), or system (system relative).

The following is a list of the names of the linkers included with CodeWarrior:

- MacOS PPC Linker
- MacOS Merge
- Java Linker
- MW JavaDoc Linker
- Win32 x86 Linker

For example, the following statement changes the linker to the Macintosh PowerPC linker:

---

```
Set Preferences of panel "Target Settings" to ¬
{Linker: "MacOS PPC Linker"}
```

---

### File Mapping Information

The File Mapping Information is a list of all the types of files you can include in the current project. It contains records described in [Table 18.10](#).

**Table 18.10 File Mapping Information Class**

Name	Property Type
File Type	string
Extension	string
Precompiled	boolean
Resource File	boolean
Launchable	boolean
Ignored by Make	boolean
Compiler	string

The Compiler string must be the name of one of the files in the Compilers folder of the CodeWarrior Plugins folder. These names are different from the names that appear in the Compiler pop-up menu of the Target preference panel. [Table 18.11](#) shows you which name to use for the compilers included with CodeWarrior.

**Table 18.11 Choosing a compiler**

To target...	with...	specify this string.
Power Macintosh	Metrowerks C/C++	"MW C/C++ PPC"
	Metrowerks Pascal	"MW Pascal PPC"
	Rez	"Rez"
	Library Importer	"Lib Import PPC"
	PEF Importer	"PEF Import PPC"
	XCOFF Importer	"XCOFF Import PPC"

To target...	with...	specify this string.
Win32/x86	Metrowerks C/C++	"MW C/C++ x86"
	Resource Compiler	"MW WinRC"
	Resource Importer	"WinRes Import"
	x86 Lib Importer	"Lib Import x86"
	x86 Obj Import	"Obj Import x86"
Java	Java	"MW Java"

To specify that a file isn't compiled, use the empty string " " for the compiler. For example, these statements show how to add an entry for text files that end in .txt and are not compiled.

---

```
set currPrefs to Get Preferences from panel "Target"
set Mappings of currPrefs to ¬
  Mappings of currPrefs & ¬
  {{File Type:"TEXT", Extension:".txt", ¬
    Compiler:"", Precompiled:false, ¬
    Resource File:false, Launchable:true, ¬
    Ignored by Make:true}}
Set Preferences of panel "Target" to currPrefs
```

---

## Segment

The Segment Class properties, as shown in [Table 18.12](#), contain information about a segment or group in the open project,

**Table 18.12 Segment Class**

Name	Property Type
name	string
filecount (read only)	small integer

## Project File

The Project File Class contains information about an entry in a project file. [Table 18.13](#) illustrates the available properties.

**Table 18.13 Project File Class**

Name	Property Type
filetype (read only)	<ul style="list-style-type: none"> <li>• source</li> <li>• unknown</li> </ul>
name (read only)	string
disk file (read only)	file specification
codesize (read only)	integer
datasize (read only)	integer
up to date (read only)	boolean
symbols	boolean
initialize before	boolean
includes (read only)	file specification
weak link (PPC only)	boolean

## Compiler Classes

- [C/C++ Compiler](#)
- [PPCAsm Assembler Class](#)
- [Java Compiler](#)
- [Pascal Compiler](#)
- [Rez Resource Compiler](#)
- [Windows Resource Compiler \(WinRC\)](#)

### C/C++ Compiler

[Table 18.14](#) lists the CodeWarrior C/C++ Compiler class properties.

**NOTE** In AppleScript, you must refer to the C/C++ Language preference panel as: panel "C/C++ Compiler".

---

**Table 18.14 C/C++ Compiler Class**

Name	Property Type
Prefix File	string
Activate CPlusPlus	boolean
ARM Conformance	boolean
ANSI Keywords Only	boolean
Require Function Prototypes	boolean
Expand Trigraph Sequences	boolean
Enums Always Ints	boolean
MPW Pointer Type Rules	boolean
Exception Handling	boolean
AutoInlining	boolean
Pool Strings	boolean
Dont Reuse Strings	boolean
ANSI Strict	boolean
MPW Newlines	boolean
RTTI	boolean
Multibyte Aware	boolean
Enable wchar_t	boolean
Use Unsigned Chars	boolean
ECPlusPlus Compatibility	boolean
Objective C	boolean

<b>Name</b>	<b>Property Type</b>
Inlining	<ul style="list-style-type: none"> <li>• inline_none</li> <li>• inline_smart</li> <li>• inlinedepth_1</li> <li>• inlinedepth_2</li> <li>• inlinedepth_3</li> <li>• inlinedepth_4</li> <li>• inlinedepth_5</li> <li>• inlinedepth_6</li> <li>• inlinedepth_7</li> <li>• inlinedepth_8</li> <li>• inline_always</li> </ul>
Enable bool Support	boolean
Direct To SOM	<ul style="list-style-type: none"> <li>• SOMoff</li> <li>• SOMon</li> <li>• SOMonWithEnv</li> </ul>
Deferred Inlining	boolean

[Table 18.16](#) lists the CodeWarrior C/C++ Warnings class properties.

### **PPCAsm Assembler Class**

This class allows you to control the features of the PowerPC assembler. [Table 18.15](#) lists the scripting commands you use to control these features.

**Table 18.15    PPCAsm Class**

<b>Name</b>	<b>Property Type</b>
PPCAsm Prefix File	string
PPCAsm Type Checking	boolean
PPCAsm Disable Warnings	boolean

Name	Property Type
PPCAsm Case Sensitive	boolean
PPCAsm Symbolic Debugging	<ul style="list-style-type: none"> <li>• sym auto</li> <li>• sym manual</li> </ul>
PPCAsm Assembler Dialect	<ul style="list-style-type: none"> <li>• dial_Power</li> <li>• dial_PowerPC</li> <li>• dial_PPC64</li> </ul>

**Table 18.16 C/C++ Warnings Class**

Name	Property Type
Unused Variables	boolean
Inconsistent Class Struct	boolean
Unused Arguments	boolean
Illegal Pragmas	boolean
Empty Declarations	boolean
Possible Errors	boolean
Extra Commas	boolean
Extended Error Checking	boolean
Treat Warnings As Errors	boolean
Hidden Virtual Functions	boolean
Implicit Arithmetic Conversions	boolean
NonInlined Functions	boolean

### Java Compiler

[Table 18.17](#) lists the CodeWarrior Java Compiler class properties.

**Table 18.17 Java Compiler Class**

Name	Property Type
Method Inlining	boolean

### Pascal Compiler

[Table 18.18](#) lists the Pascal Language Class options. (In AppleScript, you must refer to the Pascal Language preference panel as: panel "Pascal Compiler")

**Table 18.18 Pascal Compiler Class**

Name	Property Type
Activate Range Checking	boolean
Use Propagation	boolean
Activate Overflow Checking	boolean
Case Sensitive	boolean
ANS Conformance	boolean
Activate ObjectPascal	boolean
Strings copy using length byte	boolean
Pool Strings	boolean
Dont Reuse Strings	boolean
Pool Sets	boolean
Dont Reuse Sets	boolean
Prefix File	string
Relax Pointer Compatibility	boolean
Optimize class hierarchy	boolean
Pointer based objects	boolean
Expand method tables	boolean

Name	Property Type
Inline method dispatching	boolean
Activate NilChecking	boolean
Trap Unmatched Cases	boolean
Copy Value Parameter	boolean
Turbo Pascal IO	small integer

[Table 18.19](#) lists the Pascal Warnings class properties.

**Table 18.19 Pascal Warnings Class**

Name	Property Type
Modified ForLoop Indexes	boolean
Function Returns	boolean
Undefined Routines	boolean
GotoAndLabels	boolean
BranchingIntoWith	boolean
BranchingIntoFor	boolean
BranchingBetweenCase	boolean
BranchingBetweenIfAndElse	boolean
Unused Variables	boolean
Unused Arguments	boolean
Check string param sizes	boolean

### Rez Resource Compiler

[Table 18.20](#) lists the properties for the CodeWarrior Rez Compiler Class.

**Table 18.20 Rez Compiler Class**

Name	Property Type
Redeclared Types	boolean
RezPrefix File	string
Escape Control Chars	boolean
Max width	small integer
Filter Mode	Skip, or Only
Filtered Types	string
Alignment	small integer
Script Mode	Roman, Japanese, Korean, SimpChinese, or TradChinese

### **Windows Resource Compiler (WinRC)**

[Table 18.21](#) lists the properties for the Windows Resource Compiler Class.

**Table 18.21 Windows Resource Compiler Class**

Name	Property Type
Prefix File	string

## **CodeGen Classes**

- [PPC Global Optimizer](#)
- [x86 Global Optimizer](#)
- [PPC CodeGen](#)
- [IR Optimizer](#)
- [Win32/x86 CodeGen](#)

### **PPC Global Optimizer**

[Table 18.23](#) lists the PowerPC optimizer class properties.

**Table 18.22 PPC Optimizer Class**

Name	Property Type
Level	<ul style="list-style-type: none"><li>• 0</li><li>• 1</li><li>• 2</li><li>• 3</li><li>• 4</li></ul>
Optimize For	<ul style="list-style-type: none"><li>• code_Size</li><li>• code_Speed</li></ul>

### **x86 Global Optimizer**

[Table 18.23](#) lists the x86 optimizer class properties.

**Table 18.23 x86 Optimizer Class**

Name	Property Type
Level	<ul style="list-style-type: none"><li>• 0</li><li>• 1</li><li>• 2</li><li>• 3</li><li>• 4</li></ul>
Optimize For	<ul style="list-style-type: none"><li>• code_Size</li><li>• code_Speed</li></ul>

### **PPC CodeGen**

[Table 18.24](#) lists the PPC Processor preference panel class properties. In AppleScript, you must refer to the PPC Processor preference panel as: panel "PPC CodeGen"

**Table 18.24 PPC CodeGen Class**

Name	Property Type
Struct Alignment	<ul style="list-style-type: none"> <li>• Align_PPC</li> </ul>
Processor	<ul style="list-style-type: none"> <li>• PPC_Generic</li> <li>• PPC_601</li> <li>• PPC_603</li> <li>• PPC_603e</li> <li>• PPC_604</li> <li>• PPC_604e</li> <li>• PPC_750</li> </ul>
Peephole Optimizer	boolean
Use Profiler	boolean
Make String ReadOnly	boolean
Schedule	boolean
Store Vector Data in TOC	boolean
Use FMADD Instructions	boolean
Processor Specific	boolean
Traceback Tables	<ul style="list-style-type: none"> <li>• TB_None</li> <li>• TB_Inline</li> <li>• TB_OutOfLine</li> </ul>
Altivec	boolean
Generate VRSAVE	boolean
Use BuiltIn Routines	boolean
Auto-Vectorize Code	boolean

### IR Optimizer

[Table 18.25](#) lists the IR Optimizer class properties.

**Table 18.25 IR Optimizer Class**

Name	Property Type
Optimize Space	boolean
Optimize Speed	boolean
Common Subexpressions	boolean
Loop Invariants	boolean
Propagation	boolean
Dead Store Elimination	boolean
Strength Reduction	boolean
Dead Code Elimination	boolean
Lifetime Analysis	boolean
Optimizations Log	boolean

### Win32/x86 CodeGen

[Table 18.26](#) lists the x86 CodeGen class properties.

**Table 18.26 Win32/x86 CodeGen Class**

Name	Property Type
X86 Machine Code Listing	boolean
X86 Byte Alignment	small integer
X86 Sym Debug Information	boolean
X86 CodeView Debug Info	boolean
MMX ThreeDNow Convention	boolean
X86 Expand Intrinsics	boolean
Register Coloring	boolean
Extended Instructions MMX	boolean
Extended Instructions ThreeDNow	boolean

<b>Name</b>	<b>Property Type</b>
X86 Processor	<ul style="list-style-type: none"> <li>• Generic X86</li> <li>• Pentium</li> <li>• Pentium Pro</li> <li>• Pentium II</li> <li>• AMD K6</li> <li>• AMD K7</li> </ul>
X86 Exception Handling	<ul style="list-style-type: none"> <li>• Exceptions</li> <li>• Zero Overhead</li> <li>• Exceptions MS Compatible</li> </ul>

## Disassembler Classes

- [PPC Disassembler](#)

### PPC Disassembler

[Table 18.27](#) lists the properties for the PowerPC Disassembly Class.

**Table 18.27 PowerPC Disassembly Class**

<b>Name</b>	<b>Property Type</b>
Show Code	boolean
Show Source	boolean
Dont show hex	boolean
Show Data	boolean
Show Exceptions	boolean
Show SYM	boolean
Show Names	boolean
Use Extended Mnemonics	boolean

## Linker Classes

- [CFM68K Linker](#)
- [Java Linker](#)
- [Mac OS Merge Linker](#)
- [PowerPC Linker](#)
- [PowerPC PEF Linker](#)
- [Win32/x86 Linker](#)
- [Output Flags Class](#)

### CFM68K Linker

[Table 18.28](#) lists the CFM68K Linker class properties.

**Table 18.28 CFM68K Linker Class**

Name	Property Type
Export Symbols	<ul style="list-style-type: none"><li>• none</li><li>• expfile</li><li>• all</li><li>• pragma</li></ul>
Old Definition	integer
Old Implementation	integer
Current Version	integer
Share Data Section	boolean
Expand Uninitialized Data	boolean
Fragment Name	string
Initialization Name	string
Main Name	string
Termination Name	string
Force Indirect Access	boolean

Name	Property Type
Far Data Threshold	integer
Global Data Alignment	<ul style="list-style-type: none"> <li>• align1byte</li> <li>• align2byte</li> <li>• align4byte</li> <li>• align8byte</li> </ul>
Library Folder ID	small integer

### Java Linker

[Table 18.29](#) lists the Java Linker class properties.

**Table 18.29 Java Linker Class**

Name	Property Type
File Name	string
File Creator	string
	Possible values include 'JAVA' or 'MWZP' as well as the creator types for Metrowerks Java or ClassWrangler.
File Type	string
	(this is 4 characters: 'ZIP ')
Output Type	<ul style="list-style-type: none"> <li>• zip file</li> <li>• runable zip file</li> <li>• droplet</li> <li>• folder</li> </ul>
Compress Zip	boolean

### Mac OS Merge Linker

[Table 18.30](#) lists the Mac OS Merge Linker class properties.

**Table 18.30 Mac OS Merge Linker Class**

Name	Property Type
Project Type	constant (The type of the project)
File Name	boolean
File Creator	string  (The creator type of the finished binary)
File Type	string  (The file type of the finished binary)
Suppress Warnings	boolean
Copy Fragments	boolean
Copy Resources	boolean
Skip Resource Types	string
Flatten Resources	boolean

### PowerPC Linker

[Table 18.31](#) lists the PPC Linker class properties.

**Table 18.31 PPC Linker Class**

Name	Property Type
Generate SYM File	boolean
Full Path In Sym Files	boolean
Generate Link Map	boolean

<b>Name</b>	<b>Property Type</b>
Link Mode	<ul style="list-style-type: none"> <li>• fast</li> <li>• normal</li> <li>• slow</li> </ul>
Suppress Warnings	boolean
Initialization Name	string
Main Name	string
Termination Name	string
Strip Static Init Code	boolean
Duplicate Item Warning	boolean

### **PowerPC PEF Linker**

[Table 18.32](#) lists the PPC PEF class properties.

**Table 18.32 PPC PEF Class**

<b>Name</b>	<b>Property Type</b>
Export Symbols	<ul style="list-style-type: none"> <li>• none</li> <li>• expfile</li> <li>• all</li> <li>• pragma</li> </ul>
Old Definition	integer
Old Implementation	integer
Current Version	integer
Code Sorting	<ul style="list-style-type: none"> <li>• nosort</li> <li>• pragmas</li> <li>• depth</li> <li>• breadth</li> <li>• sortfile</li> </ul>
Share Data Section	boolean

---

Name	Property Type
Expand Uninitialized Data	boolean
Fragment Name	string
Library Folder ID	small integer
Collapse Reloads	boolean

---

**Win32/x86 Linker**

[Table 18.33](#) lists the x86 Linker class properties.

**Table 18.33 x86 Linker Class**


---

Name	Property Type
Generate SYM File	boolean
Entry Point Usage	<ul style="list-style-type: none"> <li>• none</li> <li>• default</li> <li>• user specified</li> </ul>
Entry Point	string
SubSystem	<ul style="list-style-type: none"> <li>• unknown</li> <li>• native</li> <li>• Windows GUI</li> <li>• Windows CUI</li> <li>• Windows CE GUI</li> </ul>
SubSystem Major Id	small integer
SubSystem Minor Id	small integer
User Major Id	small integer
User Minor Id	small integer
Generate Link Map	boolean

---

<b>Name</b>	<b>Property Type</b>
Generate CV Info	boolean
Command Line File	string

### **Output Flags Class**

These properties change the behavior of the post-linker. You can lock the output file, set resource flags in the file, and other operations.

**Table 18.34 Output Flags Class**

<b>Name</b>	<b>Property Type</b>
Output Locked	boolean
Resources Locked	boolean
Print Driver Multifinder	boolean
Invisible	boolean
Has Bundle	boolean
Name Locked	boolean
Stationery	boolean
Has Custom Icon	boolean
Shared	boolean
Initiated	boolean

Name	Property Type
Label	<ul style="list-style-type: none"><li>• no label</li><li>• Label 1</li><li>• Label 2</li><li>• Label 3</li><li>• Label 4</li><li>• Label 5</li><li>• Label 6</li><li>• Label 7</li></ul>
Comment	string

## Build Classes

- [Build Extras](#)
- [Error Information](#)

### Build Extras

[Table 18.35](#) describes the properties for the Build Extras Class.

**Table 18.35 Build Extras Class**

Name	Property Type
Browser active	boolean
Modification date caching	boolean
Dump Browser Info (read only)	boolean
Cache Subproject Data (read only)	boolean

### Error Information

This class describes a single error or warning from the compiler or the linker. This class is used by all compilers for all processors. The properties for this class are listed in [Table 18.36](#).

**Table 18.36 Error Information Class**

Name	Property Type
messageKind (read only)	<ul style="list-style-type: none"> <li>• information</li> <li>• compiler error</li> <li>• compiler warning</li> <li>• definition</li> <li>• linker error</li> <li>• linker warning</li> <li>• find result</li> <li>• generic error</li> </ul>
message (read only)	string
disk file (read only)	file specification
line Number (read only)	integer

## Browser Classes

- [Browser Coloring](#)
- [Browser Catalog](#)
- [Function Information](#)

### Browser Coloring

[Table 18.37](#) lists the **Browser Coloring** preference panel properties.

**Table 18.37 Browser Coloring Class**

Name	Property Type
Browser Keywords	boolean
Classes Color	RGB values list
Constants Color	RGB values list

---

Name	Property Type
Enums Color	RGB values list
Functions Color	RGB values list
Globals Color	RGB values list
Macros Color	RGB values list
Templates Color	RGB values list
Typedefs Color	RGB values list

---

**Browser Catalog**

The Browser Catalog Class elements may be referred to by numeric index, and by name.

**Function Information**

The Function Information class properties are described in [Table 18.38](#).

**Table 18.38** **Function Information Class Properties**


---

Name	Property Type
disk file (read only)	file specification
lineNumber (read only)	integer

---

**Editor Classes**

- [Editor](#)
- [Font](#)
- [Document](#)
- [Character](#)
- [Insertion Point](#)
- [Custom Keywords](#)
- [Line](#)

- [Text](#)
- [Selection-Object](#)
- [Syntax Coloring](#)
- [Window](#)
- [Layout Editor Class](#)

### **Editor**

[Table 18.39](#) lists the Editor class properties.

**Table 18.39    Editor Class**

Name	Property Type
Remember window	boolean
Main Text Color	RGB values list
Background Color	RGB values list
Context Popup Delay	boolean
Remember selection	boolean
Use Drag & Drop Editing	boolean
Flash delay	integer
Dynamic scroll	boolean
Balance	boolean
Remember font	boolean
Sort Function Popup	boolean
Use Multiple Undo	boolean
Save on update	boolean

An RGB values list is a list of three numbers from 0 to 65,535 that specifies how much red, green, and blue a color contains. For example, this example code sets the main text color to red.

```
set Prefs to Get Preferences from panel "Editor"
set Main Text Color of Prefs to {65535,0,0}
Set Preferences of panel "Editor" to Prefs
```

---

## Font

[Table 18.40](#) lists the Font class properties.

**Table 18.40** **Font Class**

Name	Property Type
Auto Indent	boolean
Tab size	small integer
Text font	string
Text size	small integer

## Document

This class, shown in [Table 18.41](#), contains class properties for a text file opened with the CodeWarrior Editor. The plural form for this class should be referred to as Documents.

The elements for a document are:

- character by numeric index, before/after another element, as a range of elements, or satisfying a test
- insertion point before/after another element
- line by numeric index, as a range of elements, before/after another element
- text as a range of elements

**Table 18.41 Document Class**

Name	Property Type
name (read only)	string
kind	<ul style="list-style-type: none"> <li>• project</li> <li>• editor document</li> <li>• message</li> <li>• file compare</li> <li>• catalog document</li> <li>• class browser</li> <li>• single class browser</li> <li>• symbol browser</li> <li>• class hierarchy</li> <li>• single class hierarchy</li> <li>• project inspector</li> <li>• ToolServer worksheet</li> <li>• build progress document</li> </ul>
file permissions (read only)	<ul style="list-style-type: none"> <li>• read write</li> <li>• read only</li> <li>• checked out read write</li> <li>• checked out read only</li> <li>• checked out read modify</li> <li>• locked</li> <li>• none</li> </ul>
location (read only)	file specification
index (read only)	integer
window (read only)	window

### Character

[Table 18.42](#) describes the Character class properties.

**Table 18.42 Character Class Properties**

Name	Property Type
offset (read only)	integer
length (read only)	integer

### Insertion Point

[Table 18.43](#) describes the Insertion Point Class properties.

**Table 18.43 Insertion Point Class Properties**

Name	Property Type
length (read only)	integer
offset (read only)	integer

### Custom Keywords

[Table 18.44](#) describes the Custom Keywords class properties.

**Table 18.44 Custom Keywords Class Properties**

Name	Property Type
Custom color 1	RGB color values list
Custom color 2	RGB color values list
Custom color 3	RGB color values list
Custom color 4	RGB color values list

### Line

[Table 18.45](#) describes the properties for the Line class. The plural form of the class should be referred to as Lines. This class has elements that may be described as follows:

- character by numeric index, as a range of elements, and before/after another element

**Table 18.45 Line Class Properties**

Name	Property Type
index (read only)	integer
offset (read only)	integer
length (read only)	integer

### Text

[Table 18.46](#) describes the Text class properties. The Text Class has the following elements:

- character by numeric index, before/after another element, as a range of elements
- insertion point before/after another element
- line by numeric index, as a range of elements, before/after another element
- text as a range of elements

**Table 18.46 Text Class Properties**

Name	Property Type
offset (read only)	integer
length (read only)	integer

### Selection-Object

[Table 18.47](#)describes the Selection-Object class properties. The elements of this object are:

- character by numeric index, before/after another element, as a range of elements, or satisfying a test
- line by numeric index, as a range of elements, or before/after another element
- text as a range of elements

**Table 18.47 Selection-Object Class Properties**

Name	Property Type
contents	type class
length (read only)	integer
offset (read only)	integer

### Syntax Coloring

[Table 18.48](#) describes the Syntax Coloring class properties.

**Table 18.48 Syntax Coloring Class Properties**

Name	Property Type
Syntax coloring	boolean
Comment color	RGB color values list
Keyword color	RGB color values list
String color	RGB color values list
Custom color 1	RGB color values list
Custom color 2	RGB color values list
Custom color 3	RGB color values list
Custom color 4	RGB color values list

### Window

[Table 18.49](#) describes the Window class properties. The plural form of this object is Windows.

**Table 18.49 Window Class Properties**

Name	Property Type
name	string
index	integer

Name	Property Type
bounds	bounding rectangle
document (read only)	document
position (read only)	point
visible (read only)	boolean
zoomed	boolean

### Layout Editor Class

The layout editor properties allow control over RAD tools settings.

**Table 18.50 Layout Editor for RAD Tools**

Name	Property Type
Show Component Palette	boolean
Show Object Inspector	boolean
GridSizeX	small integer
GridSizeY	small integer

## Object Classes

The object classes describe the properties of the objects in the project.

- [Member Function Class](#)
- [Base Class](#)
- [Class Class](#)
- [Data Member Class](#)

### Member Function Class

[Table 18.51](#) lists the Member Function AppleScript class properties. The plural reference to use would be Member Functions.

**Table 18.51 Member Function Class**

Name	Property Type
name (read only)	string
access (read only)	<ul style="list-style-type: none"><li>• public</li><li>• protected</li><li>• private</li></ul>
virtual (read only)	boolean
static (read only)	boolean
declaration file (read only)	file specification
declaration start offset (read only)	integer
declaration end offset (read only)	integer
implementation file (read only)	file specification
implementation end offset (read only)	integer
implementation start offset (read only)	integer

### Base Class

[Table 18.52](#) lists the Base Class AppleScript class properties. The plural reference to use would be Base Classes.

**Table 18.52 Base Class AppleScript Class Properties**

Name	Property Type
class (read only)	reference
access (read only)	<ul style="list-style-type: none"> <li>• public</li> <li>• protected</li> <li>• private</li> </ul>
virtual (read only)	boolean

### Class Class

[Table 18.53](#) lists the Class AppleScript class properties. The plural reference to use would be Classes. The elements of this class include:

- base class by numeric index
- member function by numeric index, and by name
- data member by numeric index, and by name

**Table 18.53 Class AppleScript Class Properties**

Name	Property Type
name (read only)	string
language (read only)	C C++ Pascal Object Pascal Java Assembler Unknown
declaration file (read only)	file specification
declaration start offset (read only)	integer

---

Name	Property Type
declaration end offset (read only)	integer
subclasses (read only)	list of class
all subclasses (read only)	list of class

---

**Data Member Class**

[Table 18.54](#) lists the Data Member AppleScript class properties. The plural reference to use would be Data Members.

**Table 18.54 Data Member AppleScript Class Properties**

---

Name	Property Type
name (read only)	string
access (read only)	<ul style="list-style-type: none"> <li>• public</li> <li>• private</li> <li>• protected</li> </ul>
static (read only)	boolean
declaration start offset (read only)	integer
declaration end offset (read only)	integer

---

**Miscellaneous Classes**

The Miscellaneous classes allow configuration of Version Control Systems, Extras for the project settings, the debugger, and other things.

- [Extras](#)
- [Target](#)
- [Target File](#)
- [Text Document](#)

- [Version Control System Setup](#)
- [Metronub Debugger Class](#)

Other classes are used for inheritance functions:

- [Application](#)
- [Build Progress Document](#)
- [Catalog Document](#)
- [Class Browser](#)
- [Class Hierarchy](#)
- [Editor Document](#)
- [File](#)
- [File Compare Document](#)
- [Message Document](#)
- [Project Document](#)
- [Project Inspector](#)
- [Single Class Browser](#)
- [Single Class Hierarchy](#)
- [Symbol Browser](#)
- [ToolServer Worksheet](#)

### **Extras**

[Table 18.55](#) lists the Extras class properties.

**Table 18.55    Extras Class**

Name	Property Type
Full screen zoom	boolean
External Reference	<ul style="list-style-type: none"><li>• Think Reference</li><li>• QuickView</li></ul>
Use Script Menu	boolean

---

Name	Property Type
Use Editor Extensions	boolean
Use External Editor	boolean

---

**Target**

[Table 18.56](#) lists the properties for the Target class. The plural form of Target is Targets. This class inherits all properties and elements of the given class.

**Table 18.56** Target Class

Name	Property Type
name	string
index (read only)	integer
project document (read only)	project document

---

**Target File**

[Table 18.57](#) lists the properties for the Target File class. The plural form of Target File is Target Files.

**Table 18.57** Target File Class

Name	Property Type
id (read only)	integer
type (read only)	<ul style="list-style-type: none"> <li>• library file</li> <li>• project file</li> <li>• resource file</li> <li>• text file</li> <li>• unknown file</li> </ul>
index (read only)	integer
location (read only)	file specification

---

<b>Name</b>	<b>Property Type</b>
path (read only)	string
linked (read only)	boolean
link index (read only)	integer
modified date (read only)	date
compiled date (read only)	date
code size (read only)	integer
data size (read only)	integer
debug	boolean
weak link (read only)	boolean
init before	boolean
prerequisites (read only)	list of list
dependents (read only)	list

### **Text Document**

[\*\*Table 18.58\*\*](#) lists the properties for the Text Document class. The plural form of Text Document is Text Documents.

**Table 18.58 Text Document Class**

<b>Name</b>	<b>Property Type</b>
inherits (read only)	document
modified (read only)	boolean
selection	selection-object

### **Version Control System Setup**

[\*\*Table 18.59\*\*](#) lists the VCS Setup class properties for use with version control systems.

**Table 18.59 Version Control System Class**

<b>Name</b>	<b>Property Type</b>
VCS Active	boolean
Connection Method	string
Username	string
Password	string
Auto Connect	boolean
Store Password	boolean
Always Prompt	boolean
Mount Volume	boolean
Database Path	path information
Local Root	path information

**Metronub Debugger Class**

The properties listed in Tab involve things you do with the debugger.

**Table 18.60 Metronub Properties**

<b>Name</b>	<b>Property Type</b>
Keep in Background	boolean
Auto Target Apps	boolean
Stop for 68K Traps	boolean
Catch PPC Traps	boolean
Catch 68K Traps	boolean
Download to Remote	boolean
Remote IP Address	string
Remote Port ID	small integer

Name	Property Type
Stop for PPC Traps	boolean
Always use Filemapping	boolean
Log DebugStr messages	boolean

## **Application**

The Application class elements are:

- document by numeric index, by name, and as a range of elements
- window by numeric index, by name, and as a range of elements

## **Build Progress Document**

The plural form of Build Progress Document is Build Progress Documents. This class inherits all properties and elements of the given class.

## **Catalog Document**

The plural form of Catalog Document is Catalog Documents. This class inherits all properties and elements of the given class.

## **Class Browser**

The plural form of Class Browser is Class Browsers. This class inherits all properties and elements of the given class.

## **Class Hierarchy**

The plural form of Class Hierarchy is Class Hierarchies. This class inherits all properties and elements of the given class.

## **Editor Document**

The plural form of Editor Document is Editor Documents. This class inherits all properties and elements of the given class.

## **File**

The File Class plural to use in AppleScripts is Files.

### **File Compare Document**

The plural form of File Compare Document is File Compare Documents. This class inherits all properties and elements of the given class.

### **Message Document**

The plural form of Message Document is Message Documents. This class inherits all properties and elements of the given class.

### **Project Document**

The plural form of Project Document is Project Documents. This class inherits all properties and elements of the given class.

### **Project Inspector**

The plural form of Project Inspector is Project Inspectors. This class inherits all properties and elements of the given class.

### **Single Class Browser**

The plural form of Single Class Browser is Single Class Browsers. This class inherits all properties and elements of the given class.

### **Single Class Hierarchy**

The plural form of Single Class Hierarchy is Single Class Hierarchies. This class inherits all properties and elements of the given class.

### **Symbol Browser**

The plural form of Symbol Browser is Symbol Browsers. This class inherits all properties and elements of the given class.

### **ToolServer Worksheet**

The plural form of ToolServer Worksheet is ToolServer Worksheets. This class inherits all properties and elements of the given class.

## Coding with CodeWarrior IDE and Apple Events

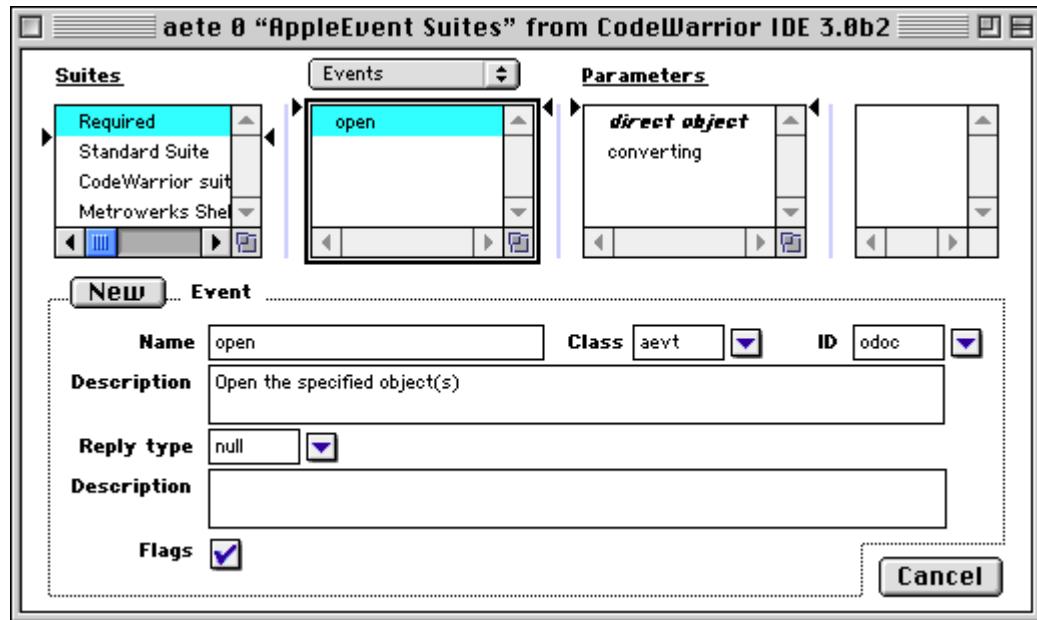
You may want to use low-level Apple Events instead of writing AppleScripts if you are producing tools or programs that need to control the CodeWarrior IDE while they are running. Third-party editors or browsers, and other tools, might require this capability.

For documentation on using low-level Apple Events in your program code, refer to *Inside Macintosh: Interapplication Communication* (Addison-Wesley) for a discussion of how to use the Apple Events portion of the Mac OS Toolbox.

There is some example code available that shows how to send Apple Events to the CodeWarrior IDE. You can find it on the CodeWarrior Reference CD in the CodeWarrior Examples folder, under the MacOS Examples folder. This code is a starter project for your work, and you will need to verify the code for proper operation. It is not intended to be a commercially-shipping product.

Largely, you will need to inspect the CodeWarrior IDE's 'aete' and 'aedt' resources using a resource editor to see what the low-level codes are to control the IDE. [Figure 18.1](#) shows an example view of what this might look like using the Resorcerer 2.0 resource editor. Rather than document all the low-level codes required to control the IDE, using a resource editor is the best solution to learn the low-level codes for now. The low-level codes may be documented at a later date.

**Figure 18.1 Resorcerer 2.0 View of the CodeWarrior IDE ‘aete’ resource**



# CodeWarrior Scripting on Microsoft Windows

---

This chapter introduces and discusses the scripting support provided by the CodeWarrior IDE on Microsoft Windows using COM, Microsoft's Common Object Model.

The sections in this chapter include:

- [CodeWarrior Windows Scripting Overview](#)
- [Tools and Reference Material](#)
- [Sample Scripts and How to Get Started](#)

---

**NOTE** The COM interfaces in the CodeWarrior IDE are a work in progress. This chapter documents the most important facets of the existing interfaces, but it is not yet exhaustive documentation.

---

## CodeWarrior Windows Scripting Overview

This chapter discusses the Microsoft Windows-based scripting facilities supported in CodeWarrior and how to begin using them. You should read this chapter if you would like to enhance and extend the capabilities of the CodeWarrior IDE.

By scripting the IDE using a scripting editor, it is possible to execute many CodeWarrior IDE commands without using the IDE directly. Scripting the CodeWarrior IDE is a way to automate repetitive tasks that do not require user interaction. There are many exciting things that you can do to harness the power of the IDE, such as automate builds, generate files automatically, and configure settings.

Scripting uses COM-based interfaces. COM is the abbreviation for Microsoft's Common Object Model. You can write scripts in any language that works with COM and runs on your computer. This includes languages such as VBScript (Visual Basic Script), JScript (ECMA-compliant JavaScript) and Perl.

This chapter's examples use VBScript. You can find examples of how to write with Perl on the CD.

---

**TIP** Look at the example scripts on the CD. Reviewing these scripts will save you time when learning to write your own.

---

This chapter is not necessarily a tutorial. If you want to learn how to edit, save, and run scripts, you may not find the information here. Instead, refer to other tools and sources of information listed in [“Tools and Reference Material” on page 732](#) for more information.

The topics in this chapter are:

- [Tools and Reference Material](#)
- [Sample Scripts and How to Get Started](#)

## Tools and Reference Material

There are several items you will want to become acquainted with in order to effectively script the IDE:

- [Microsoft Scripting Technologies Web Site](#)
- [OLE/COM Object Viewer](#)
- [Magazine Articles](#)
- [Published Books](#)

## Microsoft Scripting Technologies Web Site

You can find numerous resources and pointers to information on the world wide web at:

<http://msdn.microsoft.com/scripting/>

This address contains information about the Windows Scripting Host, the languages you can use for scripting, terminology, and debugging information.

Another useful site is:

<http://www.scripting.com>

One tool you will want to have is the Windows Scripting Host (WSH) software, available at:

<http://msdn.microsoft.com/scripting/windowshost/>

After installing WSH you can run scripts directly on your computer.

If you want to debug scripts you might find this URL to be helpful:

<http://msdn.microsoft.com/scripting/debugger/>

---

**NOTE** You can also execute scripts from the Microsoft Internet Explorer browser if the script is wrapped with the appropriate HTML tags. To learn how to do this, refer to the Microsoft scripting site for information on this and other ways to execute scripts. Note that you can download Internet Explorer at <http://www.microsoft.com/ie/>

---

## **OLE/COM Object Viewer**

You will need a copy of the OLE/COM Object Viewer application from Microsoft in order to understand how to interpret the interfaces to the CodeWarrior IDE.

You can find this application on the world wide web at:

<http://www.microsoft.com/com/resources/oleview.asp>

Other helpful editing and debugging tools may also be available from third-party vendors.

Here is a quick list of the steps you should perform to learn about using the object viewer.

1. To use OLE/COM Object Viewer, launch it and you will see something similar to [Figure 19.1](#).

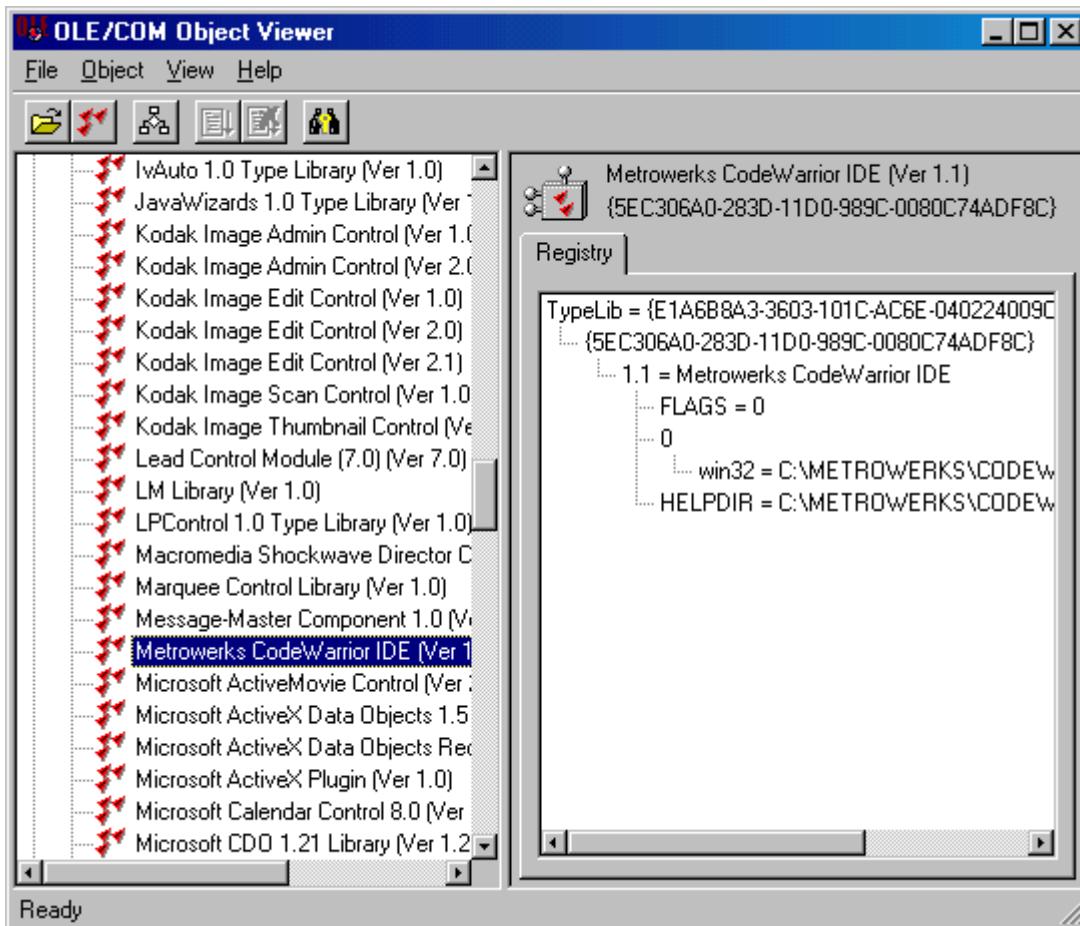
---

**NOTE** In the Visual Studio 6.0 developer tools product from Microsoft, the application is instead called **OLE Viewer**.

---

2. Use the **View** menu to set **Expert Mode**.
3. Click on the plus sign “+” next to **Type Libraries** in the left window pane.
4. Scroll down to find and click on the **Metrowerks CodeWarrior IDE** entry.
5. If you double-click on the **Metrowerks Codewarrior IDE** entry, a window like that shown in [Figure 19.2](#) appears.

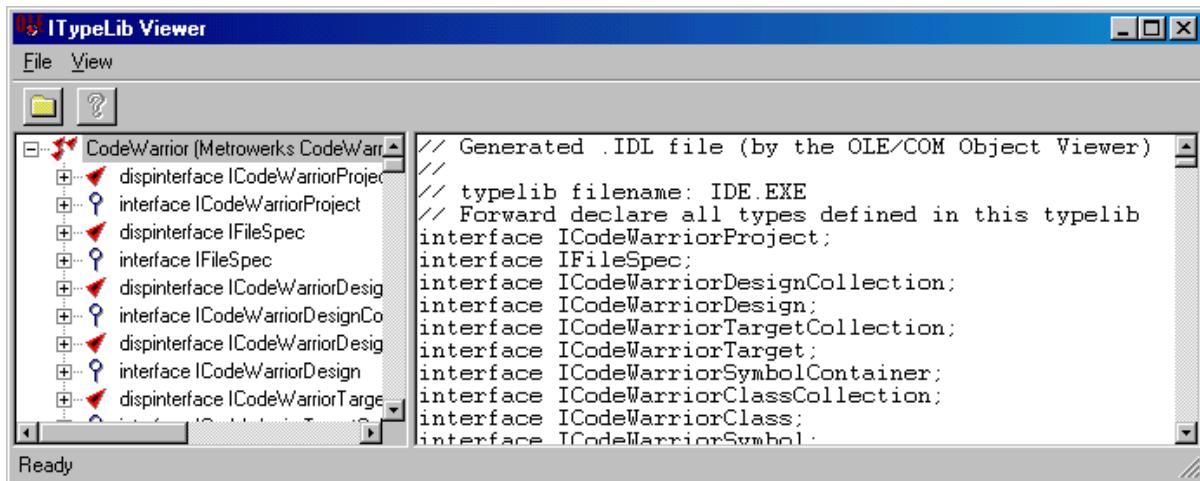
**Figure 19.1 OLE/COM Object Viewer**



**NOTE** If you don't get this window, make sure you have the latest version of `oleview.exe` from the Microsoft web site mentioned earlier in this section.

This view shows all the IDL (Interface Definition Language) for the COM objects. You can browse any particular area of interest to delve into the internals of the scripting interfaces. In particular, pay attention to coclasses, dispinterfaces, interfaces, and enums.

**Figure 19.2** ITypeLib Viewer



Using this information you can determine how to talk to the IDE to control its behavior through scripting. To learn how to do this, refer to ["An Example Script Command" on page 737](#).

## Magazine Articles

There are many articles to be found in print and on the web. One such article that is a useful introduction is titled "*Using the Windows Scripting Host*" and is published in the *Windows Developer's Journal* in October of 1999.

The web address for this magazine is:

<http://www.wdj.com>

## Published Books

You can also find books such as “*Windows Script Host Programmer's Reference*” by Dino Esposito, published by Wrox.

Another one is “*Sams Teach Yourself Windows Script Host in 21 Days*” by Thomas Fredell, Michael Morrison, Stephen Campbell (Contributor), Ian Morrish, and Charles Williams, published by Sams.

Or try “*Windows Script Host*” by Tim Hill, published by Macmillan.

If you are interested in COM, you might want to read “*Inside COM*” by Dale Rogerson, published by Microsoft Press.

## Sample Scripts and How to Get Started

This section will show you how to get a jump on starting to write your own scripts.

The topics in this section are:

- [How to Start Scripting](#)
- [A Word About Collections](#)
- [Script Examples](#)

### How to Start Scripting

This section discusses how to begin using the public interfaces of the CodeWarrior IDE to write scripts and extend the operation of the IDE.

The sections here are:

- [Creating an Application Instance](#)
- [An Example Script Command](#)

#### Creating an Application Instance

One of the first lines of your script should be something like:

---

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

---

This creates a COM instance of the CodeWarrior IDE that you can interact with in subsequent scripting operations. This is the ICodeWarriorApp class in COM.

### An Example Script Command

Now we'll analyze how you would use OLE/COM Object Viewer to learn how to script the IDE for one simple operation. In order to do much in the IDE, you will need to open a project. This can be accomplished with this script command:

---

```
set project = codewarrior.OpenProject(projectname, true, 2, 0 )
```

---

How would you know how to write this command? Simple, use the OLE/COM Object Viewer to inspect the CoClasses for the operation you want to perform, as in the steps here:

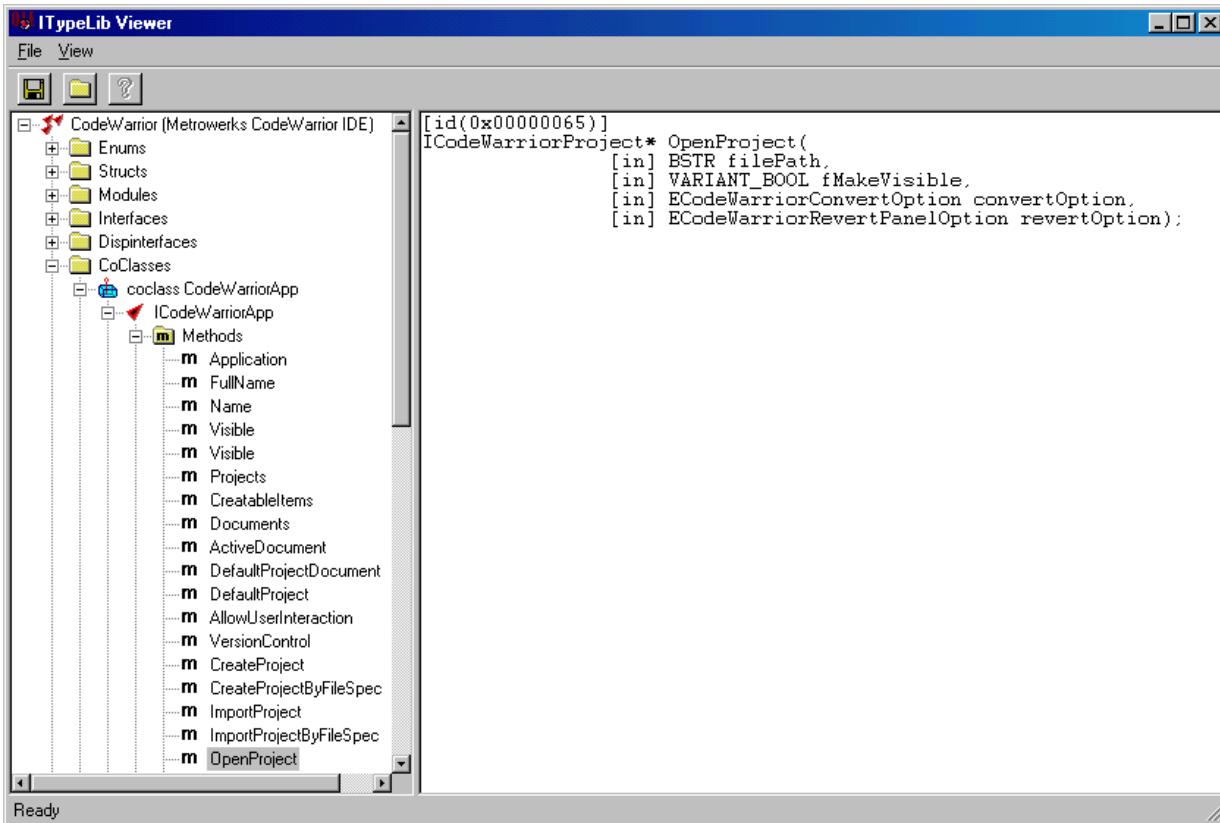
1. To do this for the `OpenProject` method, you would first launch OLE/COM Object Viewer.
2. Then use the **View** menu to put it into **Expert Mode**, click on the plus sign “+” next to **Type Libraries** in the left window pane, then double-click on the **Metrowerks CodeWarrior IDE** entry.

A window similar to [Figure 19.2 on page 735](#) appears.

3. Then use the **View** menu of the **ITypeLib Viewer** window to choose **Group by type kind**.
4. Then click on the plus sign “+” next to **CoClasses**, then the + next to **coclass CodeWarriorApp**, then the + next to **Methods**.
5. Click on **OpenProject**.

Your window should now look something like that shown in [Figure 19.3](#).

**Figure 19.3 OpenProject Method**



This shows that the `OpenProject` method requires 4 arguments, all are parameters that are passed in only (the `[in]` designation signifies this):

- `filePath` - the path to the project to be opened.
- `fMakeVisible` - whether to make the project visible or not.
- `convertOption` - whether to allow the project to be converted or not (taking a value from the `ECodeWarriorConvertOption` enumeration).
- `revertOption` - the value from the COM enumeration named `ECodeWarriorRevertPanelOption`.

You can view the values for the enumerations in the same window under the **Enums** hierarchy.

## A Word About Collections

Several of the COM interfaces use “collections.” A collection is a list of elements of some more-primitive type, such as the COM collection `ICodeWarriorComponentEventCollection`.

Collections are zero-based lists. In other words, the first element is referred to as the “zeroth element” instead of starting with the ordinal number 1 (one).

All collections share a common group of methods you can call to manipulate information for the collection, including the following:

- [Count](#)
- [ReadOnly](#)
- [Item](#)
- [Add](#)
- [Remove](#)

### **Count**

This method returns the count of the number of elements in the collection. This allows you to iterate to a specified bound since you know how many items to process in the collection.

#### ***Return Type***

The return type for `Count` is type `long`.

### **ReadOnly**

This method indicates whether the elements in the collection are modifiable or not.

#### ***Return Type***

The return type for `ReadOnly` is type `VARIANT_BOOL`.

### **Item**

This method returns the indicated element, or item, in the collection.

### ***Return Type***

The return type for Item is a pointer to the interface, as in:

---

```
[interface *] Item( [in] long index );
```

---

An example of a value for [interface\*] is  
ICodeWarriorProjectCollection if you are expecting a  
pointer to an ICodeWarriorProject.

### **Add**

This method allows you to add an element to the collection.

### ***Return Type***

The return type for Add is a pointer to the interface, as in:

---

```
Add([in] [interface*] pval );
```

---

An example of a value for [interface\*] is  
ICodeWarriorProjectCollection if you are expecting a  
pointer to an ICodeWarriorProject.

### **Remove**

This method allows you to delete an element from the collection.

### ***Return Type***

The return type for Add is a pointer to the interface, as in:

---

```
Remove([in] [interface*] pval );
```

---

An example of a value for [interface\*] is  
ICodeWarriorProjectCollection if you are expecting a  
pointer to an ICodeWarriorProject.

## **Script Examples**

Here are a couple of example scripts that illustrate the way you  
might want to write your scripts. These scripts are invoked from the

command-line (**Start/Programs/MS-DOS Prompt**) using the wscript Windows Scripting Host executable.

The scripts in this section are:

- [Removing Object Code](#)
- [Build and Wait](#)

### **Removing Object Code**

This script accepts as the command-line parameter the path to the project to be opened. The absolute path needs to be included, for example:

---

```
wscript remobj.vbs "C:\testprojects\test1.mcp"
```

---

If no command line arguments are given, the script prompts the user for the absolute path of the project file to be opened. If specified, the script tries to open the project, otherwise it opens the default one c:\testprojects\test1.mcp. This script opens the project and selects the files that belong to the default target.

#### **Listing 19.1    remobj.vbs**

---

```
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
```

## **CodeWarrior Scripting on Microsoft Windows**

### *Script Examples*

---

```
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
```

---

### **Build and Wait**

This script accepts as a command-line parameter the name of the project to be opened. The absolute path needs to be included, for example:

---

```
wscript bldwait.vbs "C:\testprojects\test1.mcp"
```

---

If no command-line arguments are given, the script prompts the user for the absolute path of the project file to be opened. If specified, the script tries to open the project, else it opens the default one "c:\testprojects\test1.mcp".

---

#### **Listing 19.2 bldwait.vbs**

---

```
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
```

---

## **CodeWarrior Scripting on Microsoft Windows**

### *Script Examples*

---

```
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
```

---

# Perl Scripting

---

You can run a Perl script as a prefix file with CodeWarrior projects. To enable this functionality, you must first install additional software plug-ins. After you install these plug-ins, you can configure CodeWarrior to recognize the Perl script.

This chapter discusses the following topics:

- [Installing the Perl Software Plug-ins](#)
- [Configuring the Perl Target Settings Panel](#)
- [Perl Scripting](#)
- [Special Considerations](#)

To learn more about Perl and Perl modules you can refer to:

---

<http://www.perl.org>

---

and

---

<http://www.cpan.org>

---

Using Perl enables the IDE to run Perl scripts as part of a project's build process. Perl scripts are executed first, before any source code compilation. This allows projects to generate source code via Perl.

## Installing the Perl Software Plug-ins

The CodeWarrior IDE uses various software plug-ins to extend its functionality. Recognizing and implementing Perl scripts in the IDE requires various Perl plug-ins. The specific plug-ins depend on the host platform you are using. The following instructions describe how to properly install the Perl plug-ins for your particular host:

- [Windows Perl Installation](#)

- [Mac OS Perl Installation](#)

## **Windows Perl Installation**

CodeWarrior implements Perl script functionality with the MWPerl.dll plug-in, the PerlDLL.dll plug-in, and the MWPerl Panel.dll plug-in. You must install all three of these plug-ins for the Perl scripts to work properly.

Place the MWPerl.dll plug-in in the Support folder. This folder resides at the following location:

---

\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Support\

---

Place the PerlDLL.dll plug-in in the Support folder. This folder resides at the following location:

---

\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Support\

---

Place the MWPerl Panel.dll plug-in within the Preference Panel folder. This folder resides at the following location:

---

\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Preference Panel

---

## **Mac OS Perl Installation**

CodeWarrior implements Perl script functionality with the MW Perl plug-in and the Perl Panel plug-in. You must install all of these components for the Perl scripts to work properly.

You will find these components in a Stuffit compressed archive on the Reference CD at:

---

Pre-Release:MW Perl Plugins:MW Perl Plugins.sit

---

Uncompress this archive, then place the MW Perl plug-in within the Compilers folder. This folder resides at the following location:

---

Metrowerks CodeWarrior:CodeWarrior Plugins:Compilers

---

Place the Perl Panel inside the Preference Panels folder.  
This folder resides at the following location:

---

Metrowerks CodeWarrior:CodeWarrior Plugins:Preference Panels

---

Finally, locate the folder called Perl5 and copy it to this location on your hard disk:

---

Metrowerks CodeWarrior:CodeWarrior Plugins:Support

---

You have now installed the required components for Mac OS Perl support.

## Configuring the Perl Target Settings Panel

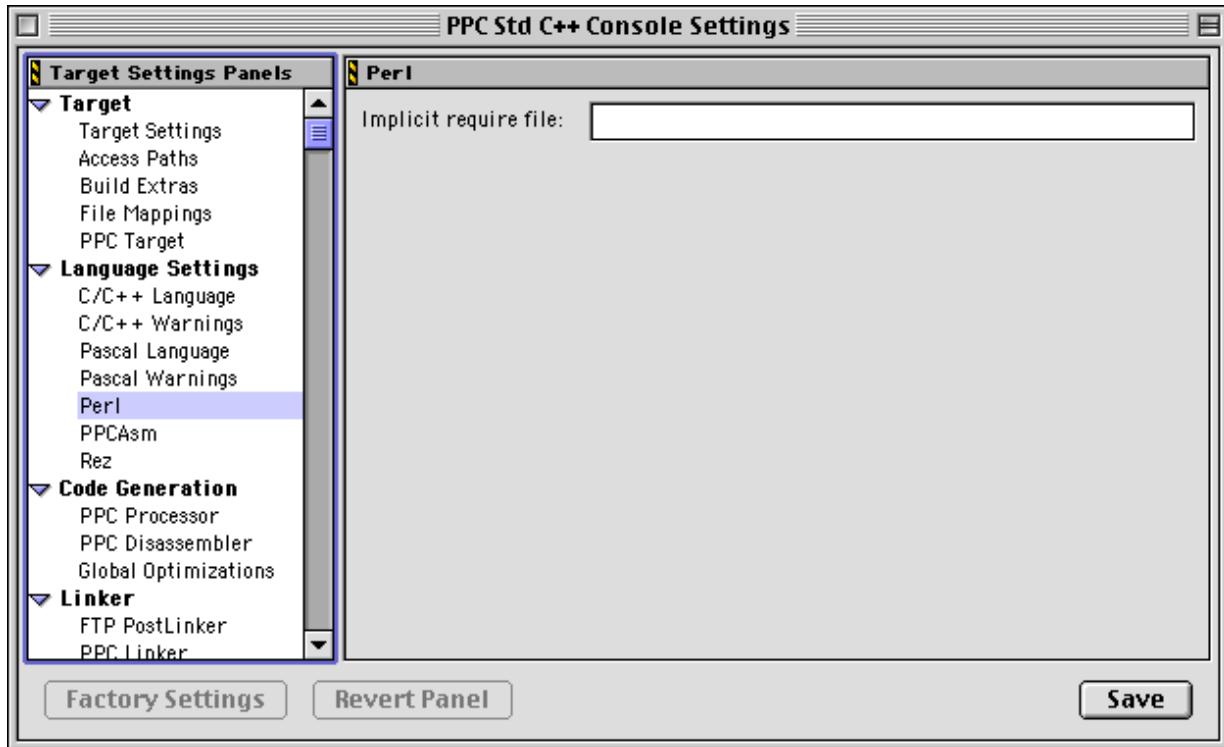
After you install the software plug-ins, CodeWarrior allows you to specify a Perl script as a prefix file for your project. In order to use the Perl script, you must first add it to your project. Then, you must configure CodeWarrior to recognize the newly added script by specifying its name in the Perl target settings panel. After you complete these settings, CodeWarrior treats the script as an implicit require file. The require directive is the Perl equivalent of the #include directive in C/C++ or the import directive in Java.

To display the Perl settings panel, choose the **Target Settings** command from the **Edit** menu. The actual name of this command depends on your currently selected build target.

## Perl Scripting

Configuring the Perl Target Settings Panel

**Figure 0.1 Perl Target Settings Panel**



Scroll through the list on the left side of the Target Settings window and highlight the **Perl** item. The Perl settings panel appears on the right side of the Target Settings window, as shown in [Figure 0.1](#).

After adding the Perl script to the project, enter the file name of the Perl script in the **Implicit require file** field. CodeWarrior treats this file as an implicit Perl `require` file when you build the project.

**NOTE** The Perl plug-in for CodeWarrior uses the find-and-load functionality of the IDE. This functionality depends on the ability of the IDE to find referenced files using absolute paths. You must specify the access paths for any files in the Perl script or the implicit require file that are not referenced via absolute paths. You specify this path information in the Access Paths settings panel. For more information, see the discussion about Access Paths in the *IDE User Guide*.

# Perl Scripting

The CodeWarrior Perl plug-in implements Perl scripting functionality in the IDE. This plug-in takes advantage of the application programming interface (API) of the IDE.

The Perl plug-in operates as a compiler in the CodeWarrior build system. After you add a Perl script to your project and compile the project, CodeWarrior runs the Perl script.

[Listing 0.1](#) shows a sample Perl script:

## **Listing 0.1    Perl Script Example**

---

```
# Simple Perl Example

# Print a line of text
print "Hello World!\n";

$scale0 = 0;
$scale1 = 1;
$scale2 = 2.5;

# Create and open a file for output
open (theFile, ">output.txt");

# Dump some text into the file
print theFile "The file should now be open\n";
print theFile "Let's try a few things:\n\n";

# Arithmetic
print theFile "***Arithmetic \n";
print theFile $scale1 + $scale2 . "\n";
print theFile $scale1 * $scale2 . "\n";
print theFile $scale1 % $scale2 . "\n\n";

# Boolean logic
print theFile "***Boolean \n";
print theFile ($scale0 && $scale0) . "\n";
print theFile ($scale0 && $scale1) . "\n";
print theFile ($scale1 && $scale1) . "\n";
```

```
print theFile ($scale0 || $scale0) . "\n";
print theFile ($scale0 || $scale1) . "\n";

print theFile (!$scale0) . "\n\n";

# Comparison
print theFile "**Comparisons \n";
print theFile ($scale2 == $scale2) . "\n";

print "That's it, closing file\n";

# Close the file
close theFile;
```

---

## Special Considerations

When using CodeWarrior to recognize and implement Perl scripts in your project, you must keep in mind some special considerations. These considerations are listed here in this section of the manual:

- [StdIn Usage](#)
- [Avoiding Link Errors](#)
- [Opening Input and Output Files](#)
- [Manipulating the IDE via Perl](#)
- [Forcing a Perl Script to Run on Every Build](#)
- [MacPerl and AppleScript](#)

### StdIn Usage

StdIn is not supported in the prefix file. This means that the Perl script cannot accept keyboard input.

### Avoiding Link Errors

The CodeWarrior linker treats the Perl script as a prefix file. Therefore, the linker expects to apply that prefix file to some source code in your project. If your project does not contain source code, the linker displays the following error message when you compile the project:

Link Error: 'main' is undefined.

Compiling the project successfully runs the script but also produces the linker error. To avoid this error message, either ensure that your project contains some source code or set the **Linker** option to **None** in the **Target Settings** preference panel. For more information, see the discussion about Target Settings in the *IDE User Guide*.

The **Link Order View** of the Project window determines when the Perl script runs. In order for your Perl script to run as intended, make sure you properly position it within this view. See the discussion about the **Link Order View** in the *IDE User Guide* for more information.

## **Opening Input and Output Files**

After you open an input and output file, the Perl script runs automatically. This situation occurs because the files create dependencies in the IDE.

## **Manipulating the IDE via Perl**

The CodeWarrior Perl interpreter features extensions that let you manipulate objects within the IDE. You can call members of the CodeWarrior Plug-in API from your Perl scripts. For more information about the Plug-in API, refer to the *IDE Plugin Manual*.

For example, you can configure the IDE to perform the following tasks via Perl scripts:

- Parse file names from a specification file and add the corresponding files to a special group within the project, for use in a later build stage.
- Play a sound between the compile stage and the link stage in the project build process.

## **Forcing a Perl Script to Run on Every Build**

Let's say you wrote a Perl script that is supposed to save files for your build. Unless the file is "touched" or altered every time you build, it may not run.

When you “touch” a file you are telling the IDE that the file’s state has changed. If you modify the file, the IDE will also see that the file has changed. However, once you have the script perfected, it won’t change again. So when you do a build, the Perl file is not executed.

To put the problem another way, when you initially run the IDE’s **Make** command, the Perl script is touched. If you don’t change the Perl script before you run the **Make** command again, the script won’t execute!

To get around this issue, and make sure that the Perl script is executed every time a build is done, you might have the Perl script touch itself for every build using one of these example methods:

- [Touch on Last Line](#)
- [Using utime\(\)](#)

### **Touch on Last Line**

The last line of your Perl script could be a system call that invokes this command:

---

```
touch thisfile.pl
```

---

This would make the Perl script named `thisfile.pl` touch itself so that the file looks changed.

### **Using utime()**

This example uses the Perl `utime( )` function to touch the script file. Include these lines in your script to force the file to be touched.

---

```
$now = time;  
utime( $now, $now, "thisfile.pl" );
```

---

## **MacPerl and AppleScript**

Note that if you are using Mac OS, MacPerl can call AppleScripts. To learn more about MacPerl, visit:

<http://www.scriptweb.com>





# Frequently Asked Questions (FAQ)

---

This chapter discusses some of the FAQs that come up when developers start working with the API and script capabilities for extending the IDE.

---

**NOTE** This section of the manual will be expanded in the future to include more FAQs, so feel free to request that your FAQ be answered here by sending email to [wordwarrier@metrowerks.com](mailto:wordwarrier@metrowerks.com)

---

## Questions and Answers

**Q.** When I select my panel in the Target Settings window, it sticks (the window locks onto that panel).

**A.** You probably have a resource ID defined in your code that doesn't exist in your project resources. This usually occurs when calling `PutData()` and you are setting data in a resource item that doesn't exist or is not numbered correctly in your resource file.

**Q.** When I choose a panel all I see is a grey field.

**A.** If you don't have the correct resource dialog ID set in your resource file as matching the one in the `initdialog()` function, then the item won't be drawn at all.

**Q.** How do I create resource files that can be shared on all Metrowerks platforms?

**A.** Use the resource flattener panel on the Cac to create a binary file for use on Windows and Unix.

## **Frequently Asked Questions (FAQ)**

### *Questions and Answers*

---

**Q.** Do I have to have a Mac to use Constructor?

**A.** Yes, you cannot edit Constructor files on any platform but a Macintosh.

**Q.** What does kCompReentrant mean? Is this supported on Windows?

**A.** Yes, but your compiler plug-in must be re-entrant and use no global variables. Each source file must have its own object file.

**Q.** Do I need to write an importer to use a linker command file from the project manager?

**A.** That depends. You can either write your own importer DLL or just add the file mappings to the linker and parse the files out from the project manager. In this way the file mappings are hard coded to the linker. Creating your own separate DLL allows the user to create custom file mappings if desired.

# Index

---

## Symbols

'\*\*\*\*' 208  
'aete' 387  
'aete' editors 388  
'cfrg' resource 61, 642  
'CTYP' 387  
'Dhlp' code resource 59  
'Dhlp' resource 272, 274, 642  
'DITL' resources 384  
'Drop' code resource 59  
'Drop' resource 643  
'EMap' Resource 269  
'EMap' resource 644  
'Flag' resource 68, 70, 202, 206, 645  
'Fmly' resource 650  
'PBkg' 386  
'PPob' 419  
'PPob' resources 384  
'STR' resource 71, 652  
'STR#' resource 72, 652  
'Targ' Resource 270  
'Targ' resource 650  
  \_stdcall 58, 75  
'PEdT' 385

## Numerics

68k code generation 700  
68K CodeGen 700

## A

A4-based code resource 59  
access path subdirectories  
  enumerating 90  
Access Paths 688  
access paths 110, 160  
  counting 90, 115  
  enumerating 90  
  getting information 114  
  subdirectories, enumerating 116  
Access Paths class 688  
accessing files  
  files open in IDE 137, 138

accessing handle memory 83  
acess paths  
  iterating 114  
Add Files 662  
adding  
  file mappings to project 207  
  project files 88, 226  
advanced request 532  
AEDisposeDesc 398  
aedt 28  
aete 28, 730  
alert  
  displaying 105  
allocating  
  handles 84  
  pointers 81  
allocating memory  
  handles 107  
  permanent 107  
  pointer 106  
API  
  changes for VCS version 8 35  
  changes for version 11 34  
  VCS version 7 changes 523  
  VCS version 8 changes 523  
  VCS version 9 changes 523  
API version  
  determining at runtime 64  
API versions 31  
appending linker output 234  
Apple Events 397  
  changing settings 397, 499  
  constructing from relative path 466  
  extracting relative path 461  
  getting settings 397, 499  
AppleEvents 537  
AppleScript 28, 383, 387  
  dictionary 387  
  terminology editors 388  
  terminology resource 387  
Application 727  
assembler  
  PowerPC 695  
associating file types with plug-ins 206

- 
- automatic library targeting 47
  - automatic precompilation 259
  - autoscrolling 395
  - auto-target libraries 47
  - avoiding link errors
    - in Perl scripts 750
  - B**
    - background drawing attachment 386
    - balancing
      - calls to load and free text 139
      - calls to lock and unlock a handle 148
      - handle allocation and disposal 112
      - pointer allocation and disposal 113
    - Balloon Help
      - adding to panels 387
    - Base Class class 720
    - base items 350
      - determining 433
    - baseItems 489
    - big-endian 360
    - binary format
      - plug-in 57, 58
    - breakpoints
      - IDE not stopping for 52
    - Bring Up To Date command 213
    - Browse Header 574
    - browse information
      - debugging 49
      - example dump 49
    - BROWSE\_EARLIEST\_COMPATIBLE\_VERSION 5 87
    - BROWSE\_HEADER 588
    - BROWSE\_VERSION 588
    - browseCompSymbolStart 222, 590
    - BrowseHeader 585
    - Browser Catalog 712
    - Browser Classes 711
    - Browser Coloring 711
    - browser data
      - class member list 582
      - fields for all records 576
      - fields for classes 580
      - fields for functions 579
      - fields for templates 584
      - function IDs 583
      - matching functions and methods 582
    - storing 227, 282
    - Browser Data Stream 575
    - browser information
      - file IDs 247
    - browser symbol dump 49
    - browser symbol type names 277
    - browser symbol types
      - custom 222
    - browser symbols
      - type names entry point 209
    - browser symbols, generating
      - determining enabled symbol types 243
    - bug fixes 30
    - Build 667
    - build 33
    - Build classes 710
    - Build Extras class 710
    - build operations
      - identifying and comparing 243
    - build progress 33
    - Build Progress Document class 727
    - byte swapping 342
  - C**
    - C/C++ Compiler class 693
    - C/C++ Warnings class 695
    - caching
      - determining if enabled 230
      - precompiled header 240
    - caching file data 230
    - calling conventions
      - Pascal 58, 75
      - plug-in entry point 58
    - canceling 96
    - canFactory 366, 489
    - canRevert 366, 489
    - cantDisassemble 203, 295
    - capability flags
      - VCS 536
    - Catalog Document class 727
    - CFM 58, 59
      - 68K 60
    - CFM code fragment 642
    - CFM68K Linker class 704
    - changes for debugging 364
    - changing input focus 373

- 
- Character class 715  
 Check 667  
 Check Syntax 668  
 Check Syntax command 213  
 checkbox 432  
     setting value 445  
 checkboxes  
     handling item hits on Mac OS 391  
 checkFileLocation 121  
 checkout states 545  
 choosing a relative path 421  
 Class Browser class 727  
 Class Class class 721  
 Class Hierarchy class 727  
 Close 681  
 Close Project 663  
 Close Window 663  
 Code Fragment Manager 58  
 code resource  
     'Dhlp' 59, 643  
     'Drop' 59, 643  
     A4-based 59  
 CodeGen class, code generation class 699  
 codes  
     operating system 208  
     processor 208  
     wildcard 208  
 CodeView 223  
 CodeWarrior  
     installation 36  
 CodeWarrior User's Guide 388  
 COM 26, 28  
 COM routines 104  
 combo box  
     getting items 433  
     inserting list items 439  
     removing items 422  
     setting items 446  
 combo box initialization  
     Windows 354  
 command line arguments 218, 288  
 command line options 244  
 command line tool 236  
 command line tools 229  
     adapting 255  
     API support 35  
     dependencies 268  
     Compile 669  
     Compile command 213  
     compile errors  
         reporting 95  
     Compile File 669  
     compiler  
         on-screen name 71  
         target platform 208, 209  
     Compiler Classes 693  
     compiling 211  
         getting current file index 248  
         getting current file location 249  
         getting file text 250  
     Component Object Model 26  
     components  
         of projects 86  
     context parameter 75, 77  
     ContextHelpAttachment 386, 387  
     control handle  
         getting 455  
     control IDs 380  
     control titles  
         getting 429  
     control values  
         getting 371, 431  
     Count 681  
     CPU, target  
         reporting 270  
     Create Project 664  
     cross-platform development  
         core API differences 61  
         differences 383  
         issues 40, 331  
         platform-dependent types 62  
         settings panel API differences 337  
     currentPrefs 344, 397, 488  
     custom browser symbol types 222  
     custom browser symbols 271  
     custom dialog items  
         activating and deactivating, Mac OS 393  
         drawing, Mac OS 392  
         highlighting, Mac OS 393  
         tracking active item 393  
     Custom Keywords class 716  
     custom symbol type names 277  
     CW\_STRICT\_DIALOGS 481, 488, 494, 518  
     cwAccessAbsolute 296
-

---

cwAccessFileName 296  
cwAccessFileRelative 297  
CWAcessPathInfo 62, 154  
CWAcessPathListInfo 155  
cwAccessPathRelative 296  
CWAcessPathType 156  
CWAddProjectEntry 88, 104, 226  
CWAddr64 157  
CWAlert 96, 105  
CWAllocateMemory 81, 106, 231  
CWAllocMemHandle 84, 107  
CWAllowV1Compatibility 528, 543  
CBrowseOptions 274  
CWCachePrecompiledHeader 230, 240, 260  
CWCompilerBrSymbol 271, 276  
CWCompilerBrSymbolInfo 271, 276  
CWCompilerBrSymbolList 277  
CWCompletionRatio 548  
CWCreateNewTextDocument 95, 108, 212  
CWDataType 92, 157  
CWDependencyInfo 62, 278  
CWDependencyType 158  
CWDialo 481  
CWDisplayLines 95, 241  
CWDonePluginRequest 77, 98, 109, 149, 529  
CWDoVisualDifference 547  
CWDROPINCOMPILERTYPE 185  
CWDROPINLINKERTYPE 185  
CWDROPINPREFSTYPE 186  
CWDROPINPREFSTYPE\_1 186  
CWDROPINVCSTYPE 187  
cwErrCantSetAttribute 194  
cwErrFileNotFound 194  
cwErrInvalidCallback 194  
cwErrInvalidMPCallback 195  
cwErrInvalidParameter 195  
cwErrObjectFileNotStored 254, 328  
cwErrOSError 118, 195  
cwErrOutOfMemory 196  
cwErrRequestFailed 78, 196  
cwErrSilent 196  
cwErrStringBufferOverflow 197  
cwErrUnknownFile 197  
cwErrUnknownSegment 328  
cwErrUserCanceled 97, 197  
CWExtensionMapping 279  
CWExtMapList 280  
CWFamily 73  
CWFamilyList 73  
CWFileInfo 62, 159  
CWFileName 162  
CWFileSpec 34, 87, 89, 162  
CWFileStateChanged 35, 523, 534, 544  
CWFileTime 35, 164, 523  
cwFileTypePrecompiledHeader 88, 187  
cwFileTypeText 88, 187  
cwFileTypeUnknown 88, 188  
CWFIndAndLoadFile 88, 110, 139, 158, 228, 229, 240  
    and caching 230  
CWFrameworkInfo 165  
CWFreeMemHandle 112, 126, 130  
CWFreeMemory 81, 113, 231  
CWFindObjectData 177, 214, 219, 228, 242  
CWGetAccessPathInfo 90, 113  
CWGetAccessPathListInfo 90, 114  
CWGetAccessPathSubdirectory 90, 115  
CWGetAPIVersion 64, 116  
CWGetArraySettingElement 369, 408, 413, 417  
CWGetArraySettingSize 409, 410, 417  
CWGetBooleanValue 369, 411  
CWGetBrowseOptions 222, 242, 275  
CWGetBuildSequenceNumber 243  
CWGetCallbackOSError 117  
CWGetCOMApplicationInterface 118  
CWGetCOMDesignInterface 119  
CWGetCommandLineArgs 34, 218, 236, 244, 288  
CWGetCommandStatus 528, 551  
CWGetComment 544  
CWGetCOMProjectInterface 118  
CWGetCOMTargetInterface 119  
CWGetDatabaseConnectionInfo 551  
CWGetEnvironmentVariable 35, 236, 244  
CWGetEnvironmentVariableCount 35, 236, 246  
CWGetFileInfo 34, 87, 88, 89, 120, 214, 219, 251  
CWGetFileText 88, 121, 139, 213  
CWGetFloatingPointValue 369, 411  
CWGetIDEInfo 64, 124  
CWGetIntegerValue 369, 412  
CWGetMainFileID 247  
CWGetMainFileNumber 213, 219, 248

---

---

CWGetMainFileSpec 213, 249  
CWGetMainFileText 213, 249  
CWGetMemHandleSize 85, 124  
CWGetModifiedFiles 250  
CWGetNamedPreferences 92, 125  
CWGetNamedSetting 368, 413, 417  
    differences in behavior during read and write 368  
CWGetOutputFileDirectory 89, 126, 223  
CWGetOverlay1FileInfo 90, 126  
CWGetOverlay1GroupInfo 90, 127  
CWGetOverlay1GroupsCount 90, 128  
CWGetOverlay1Info 90, 129  
CWGetPluginData 92, 129, 159, 540  
CWGetPluginRequest 76, 130, 149, 345, 528  
CWGetPrecompiledHeaderSpec 251, 259  
CWGetProjectFile 34, 89, 131  
CWGetProjectFileCount 89, 121, 132, 214  
CWGetProjectFileSpecifier 549  
CWGetRelativePathValue 369, 414  
CWGetResourceFile 252  
CWGetSegmentInfo 90, 132  
CWGetStoredObjectFileSpec 228, 253, 255  
CWGetStringValue 369, 415  
CWGetStructureSettingField 369, 409, 413, 416  
CWGetSuggestedObjectFileSpec 254  
CWGetTargetDataDirectory 89, 94, 133  
CWGetTargetInfo 217, 235, 255  
CWGetTargetName 89, 134  
CWGetTargetStorage 230, 256, 266  
CWGetVCSItem 534, 552  
CWGetVCSItemCount 534, 552  
CWGetVCSPointerStorage 554  
CWGetWorkingDirectory 35, 236, 257  
CWHelpInfo 482  
CWIDEInfo 165  
cwInterfaceDependency 111, 158, 283  
CWIsAdvancedRequest 532, 549  
CWIsAutoPrecompiling 212, 258  
CWIsCachingPrecompiledHeaders 230, 241, 260  
CWIsCommandSupportedRequest 532, 549  
CWIsGeneratingDebugInfo 222, 260  
CWIsPrecompiling 261  
CWIsPreprocessing 262  
CWIsRecursiveRequest 532, 549  
CWLoadObjectData 177, 214, 219, 228, 242, 255, 263  
CWLockMemHandle 83, 126, 130, 134  
CWMacOSErrToCWResult 98, 136  
CWMemHandle 82, 166  
CWMessageRef 62, 167  
CWNewProjectEntryInfo 168  
CWNewTextDocumentInfo 170  
cwNoDependency 158  
cwNoErr 197  
cwNormalDependency 111, 158  
CWObjectData 62, 281  
CWOSAlert 97  
CWOSErrorMessage 97  
CWOSResult 170  
CWOverlay1FileInfo 171  
CWOverlay1GroupInfo 171  
CWOverlay1Info 172  
CWPanelActivateItem 373, 418  
CWPanelAppendItems 378, 419  
CWPanelChooseRelativePath 35, 420  
CWPanelDeleteListItem 420, 422  
CWPanelEnableItem 372, 423  
CWPanelGetCurrentPrefs 343, 423  
CWPanelGetDebugFlag 364, 424  
CWPanelGetDialogItemHit 425  
CWPanelGetFactoryPrefs 343, 426  
CWPanelGetData 427  
CWPanelGetItemMaxLength 373, 428  
CWPanelGetItemText 371, 373, 429  
CWPanelGetItemTextHandle 371, 430  
CWPanelGetItemValue 371, 373, 392, 431  
CWPanelGetListListItemText 419, 432  
CWPanelGetNumBaseDialogItems 426, 433  
CWPanelGetOriginalPrefs 343, 434  
CWPanelGetPanelPrefs 435  
CWPanelGetRelativePathString 436  
CWPanelInsertListItem 420, 438  
CWPanelInvalItem 374, 439  
CWPanelSetFactoryFlag 366, 427, 440  
CWPanelSetItemData 441  
CWPanelSetItemMaxLength 373, 441  
CWPanelSetItemText 371, 373, 442  
CWPanelSetItemTextHandle 371, 443  
CWPanelSetItemValue 371, 373, 392, 444  
CWPanelSetListItemText 419, 445  
CWPanelSetRecompileFlag 361, 446

---

CWPanelSetRelinkFlag 361, 447  
CWPanelSetReparseFlag 448  
CWPanelSetResetPathsFlag 361, 449  
CWPanelSetRevertFlag 366, 449  
CWPanelShowItem 372, 450  
CWPanelValidItem 374, 451  
CWPanlActivateItem 452  
CWPanlAppendItems 452  
CWPanlChooseRelativePath 452  
CWPanlDrawPanelBox 392, 452  
CWPanlDrawUserItemBox 453  
CWPanlEnableItem 453  
CWPanlGetArraySettingElement 454  
CWPanlGetArraySettingSize 454  
CWPanlGetBooleanValue 454  
CWPanlGetFloatingPointValue 454  
CWPanlGetIntegerValue 455  
CWPanlGetItemControl 455  
CWPanlGetData 456  
CWPanlGetItemMaxLength 456  
CWPanlGetItemRect 374, 392, 456  
CWPanlGetItemText 457  
CWPanlGetItemTextHandle 457  
CWPanlGetValue 457  
CWPanlGetMacPort 392, 458  
CWPanlGetNamedSetting 458  
CWPanlGetPanelPrefs 458  
CWPanlGetRelativePathString 459  
CWPanlGetRelativePathValue 459  
CWPanlGetStringValue 459  
CWPanlGetStructureSettingField 459  
CWPanlInstallUserItem 460  
CWPanlInvalItem 460  
CWPanlReadBooleanSetting 460  
CWPanlReadFloatingPointSetting 460  
CWPanlReadIntegerSetting 461  
CWPanlReadRelativePathAEDesc 461  
CWPanlReadRelativePathSetting 462  
CWPanlReadStringSetting 462  
CWPanlRemoveUserItem 462  
CWPanlSetBooleanValue 462  
CWPanlSetFloatingPointValue 463  
CWPanlSetIntegerValue 463  
CWPanlSetItemData 463  
CWPanlSetItemMaxLength 463  
CWPanlSetItemText 464  
CWPanlSetItemTextHandle 464  
CWPanlSetItemValue 464  
CWPanlSetRelativePathValue 464  
CWPanlSetStringValue 464  
CWPanlShowItem 465  
CWPanlValidItem 465  
CWPanlWriteBooleanSetting 465  
CWPanlWriteFloatingPointSetting 465  
CWPanlWriteIntegerSetting 466  
CWPanlWriteRelativePathAEDesc 466  
CWPanlWriteRelativePathSetting 467  
CWPanlWriteStringSetting 467  
CWPLUGIN\_ENTRY 58, 98  
CWPlugin\_GetDefaultMappingList Entry Point 269  
CWPlugin\_GetDisplayName 71, 339  
CWPlugin\_GetDisplayName Entry Point 151  
CWPlugin\_GetDropInFlags 34, 64, 65, 68, 77, 339, 340, 484  
    differences for settings panels 339  
CWPlugin\_GetDropInFlags Entry Point 150  
CWPlugin\_GetDropInName 70, 339  
CWPlugin\_GetDropInName Entry Point 151  
CWPlugin\_GetHelpInfo 339, 376  
CWPlugin\_GetHelpInfo Entry Point 480  
CWPlugin\_GetPanelList 72  
CWPlugin\_GetTargetList Entry Point 270  
CWPluginContext 99, 172, 332  
CWPostDialog 136, 541  
CWPostFileAction 88, 137, 534, 541  
CWPreDialog 138, 541  
CWPreFileAction 88, 138, 534, 541  
CWProjectFileInfo 34, 62, 87, 174  
CWProjectSegmentInfo 178  
CWPutResourceFile 234, 264  
CWReadBooleanSetting 367, 467  
CWReadFloatingPointSetting 367, 468  
CWReadIntegerSetting 367, 469  
CWReadRelativePathSetting 367, 470  
CWReadStringSetting 367, 470  
CWRadioButton 34, 90, 178  
CWRadioButtonFormat 180  
CWRadioButtonTypes 181  
CWReleaseFileText 88, 111, 122, 139  
CWRemoveProperty 139

---

- 
- C**  
 CWReportMessage 95, 140, 541  
 CWResizeMemHandle 85, 141  
 CWResolveRelativePath 34, 90, 142  
 CWResult 182  
 CWSetBooleanValue 369, 471  
 CWSetCommandDescription 550  
 CWSetCommandStatus 528, 552  
 CWSetFloatingPointValue 369, 472  
 CWSetIntegerValue 369, 474  
 CWSetModDate 88, 143, 227  
 CWSetPluginOSSError 98, 144  
 CWSetRelativePathValue 369, 475  
 CWSetStringValue 369, 476  
 CWSetTargetInfo 217, 256, 265  
 CWSetTargetStorage 230, 256, 266  
 CWSettingID 482  
 CWSetVCSItem 35, 523, 534, 553  
 CWSetVCSPointerStorage 554  
 CWShowStatus 95, 145  
 CWStoreObjectData 158, 213, 226, 227, 229, 254,  
     255, 263, 267  
 CWStorePluginData 91, 145, 159, 540  
 CWSUCCESS 97  
 cwSystemPath 115, 156, 188  
 CWTarFileInfo 62, 217, 283  
 CWTarTargetList 289  
 CWUnlockMemHandle 83, 147  
 CWUnmangleInfo 210, 290  
 CWUserBreak 96, 148  
 cwUserPath 115, 156, 188  
 CWVCSItem 35  
 CWVCSStateChanged 35, 523, 545  
 CWVCSVersionData 35, 523  
 CWWriteBooleanSetting 367, 477  
 CWWriteFloatingPointSetting 367, 477  
 CWWriteIntegerSetting 367, 478  
 CWWriteRelativePathSetting 367, 479  
 CWWriteStringSetting 367, 479
- D**
- data  
     storing with a file 146  
 Data Member class 722  
 database connection and disconnection 527  
 dataversion 341, 484
- Debug command 218  
 debug helper 218, 235  
 debugging  
     changing settings data 364  
     determining if enabled 425  
     generating data 222  
     host application 47  
     options 48  
 debugging information 282  
     determining if enabled 261  
     generating 222  
 debugging setup 46  
     message logging 51  
 debugHelperIsRegKey 288  
 debugHelperName 288  
 debugOn 364, 490  
 DebugStr messages 51  
 dependencies  
     committing 228  
     establishing 268  
     specifying 111  
     specifying multiple 229  
 dependency information 278  
 dependency tracking 227, 229  
 dependency types 158  
 dependencyType 111  
 detailed error information 97  
 diagnostics  
     plug-in 48  
 dialog 488  
 dialog controls  
     setting title as handle 444  
 dialog items  
     cancelling redraw 451  
     framing 453  
     getting control from 455  
     getting item rectangle 457  
     getting port 458  
     number of base items 433  
     showing 450  
 dialog manager, Mac OS 494  
 DialogPtr 494  
 dialogs  
     operating system, displaying 96, 137, 138  
 dictionary editors 388  
 dictionary resources 387  
 disabling panel items 372, 423
-

- 
- Disassemble File 670
  - Disassembler classes 703
  - disassembling 219
  - disassembly
    - displaying output 220
  - display name 70
    - entry point 152
  - displaying compilation status 95
  - displaying status 95
  - displaying text 109
  - disposing
    - handles 84, 112
    - pointers 81, 113
  - disposing text 139
  - DLL 58
  - Document class 714
  - doPrecompile 645
  - DOS wildcard 421
  - drag and drop 383, 395
    - dragging an item 396
    - ending a drag 396
    - exiting a drag 396
    - rejecting a drag 395
    - scrolling 395
    - starting a drag 395
  - Drag Manager Programmer's Guide 396
  - dragmouse 490
  - dragrect 395, 490
  - dragref 395, 490
  - dropin name 70
    - entry point 151
  - DROPINCOMPILERLINKERAPIVERSION 297
  - DropInFlags 66, 80, 182, 225
  - dropinflags 340, 484
  - DROPINPANELAPIVERSION 494
  - dropintype 340, 483
  - duplicating the IDE 46
  - DWARF 223
  
  - E**
  - EAccess 588
  - earliestCompatibleAPIVersion 68, 340, 484, 536
  - EBrowserItem 209, 589
  - ECMA 28
  - edit fields
    - determining maximum length 428
    - getting and setting text 371
    - getting as handle 430
    - getting as numbers 431
    - getting numerical values from 372
    - getting text 429
    - setting 443
    - setting to a handle 444
    - settings maximum length 442
    - validating content 373
  - Edit menu
    - determining item status 394
    - item handling 393
    - responding to commands 394
  - edit text 432
    - setting value 445
  - Editor class 713
  - Editor Classes 712
  - Editor Document class 727
  - editor window
    - creating 95, 109
  - ELanguage 590
  - EMember 591
  - enabling panel items 372, 423
  - end
    - build 33
  - entry point
    - browser symbol 271
    - browser symbol type names 209, 642
    - CWPlugin\_GetDropInFlags 34, 64
    - CWPlugin\_GetDropInFlags (preference panel) 68
    - display name 152
    - dropin name 151
    - file mappings 206, 269
    - help information 375, 481
    - linker 210
    - main 75
    - plug-in
      - flags 150
    - settings panel capabilities 340
    - symbol unmangling 210, 273, 642
    - target list 270
  - entry points
    - compiler and linker 206
    - informational 65
    - settings panel 339
    - VCS 535
  - enumerating subdirectories 116

- 
- environment variables 218, 236, 245
    - counting 246
  - error codes
    - converting Mac OS to IDE 136
    - returning to IDE 77, 98, 149
    - settings panels 351
  - error codes, operating system
    - reporting to IDE 98
  - error information
    - obtaining 97
  - Error Information class 710
  - error result
    - returning 109
  - errors
    - adding files to a project 51
    - displaying 141
    - duplicate plug-ins 52
    - IDE error 2 52
    - opening test project 44
    - selecting a compiler 52
  - errors, operating system
    - returning to IDE 145
  - errors, reporting 95
  - ETemplateType 592
  - event 489
  - EventRecord 391
  - executable
    - linkage model 287
    - plug-in 57
  - Execution Mode
    - VCS 555
  - exelinkageFlat 297
  - exelinkageOverlay1 298
  - exelinkageSegmented 298
  - Export Project command 366
  - exporting settings data 366
  - external plug-in data 94
  - Extras class 723
  
  - F**
  - factory flag
    - setting 366
  - factoryPrefs 344, 488
  - families
    - settings panel 650
  - family
    - panel 73
  - family name
    - matching 74
  - fat plug-in 58
  - file
    - adding to project 226
    - retrieving 88
  - file actions, pre and post 524
  - File class 728
  - File Compare Document class 728
  - file data
    - storing 146
  - file format
    - SYM and xSYM 223
  - file IDs
    - in browser information 247
  - file information
    - obtaining 87, 120
  - file lists, VCS 534
  - file mapping entry point 269
  - File Mapping Information class 690
  - file mappings 280
    - adding to project 207
    - resource 644
    - specifying 206
    - when added to project 269, 645
  - File Mappings panel 269
  - file name
    - extension 644
  - file specification 87
    - for project file 132
    - getting 120
  - file type codes 269
  - file types
    - associating with plug-ins 269
  - filedatatype 88
  - files
    - loading contents of 88
    - specifying 87
  - FileSpec
    - Mac OS 87
  - filterList 421
  - finding files 110
  - flags
    - file mapping 644
  - Font class 714
  - forcing
    - panel item redraw 440

project file rescan 361  
project recompilation 361  
target relinking 361  
four-character code 157  
framing panel items 453  
freeing  
  handles 112  
  pointers 113  
freeing text 139  
Function Information class 712

**G**

generating debugging information 222  
Get 675  
Get Definition 675  
Get Member Function Names 676  
Get Nonsimple Classes 676  
Get Open Documents 676  
Get Preferences 676  
Get Project File 678  
Get Project Specifier 679  
Get Segments 680  
getting 89  
getting panel data 356  
getting project information 89  
getting started 36  
getting target information 89  
global data  
  maintaining 230  
  storing 266  
global storage  
  disposing 234  
globals  
  and threading 219  
Goto Function 682  
Goto Line 682  
GrafPort  
  getting 458  
GrafPtr 392

**H**

handle allocation 107  
handle memory 82  
handle relocation 83  
handle size  
  determining 125

handledByMake 645  
handles 166  
  accessing 83  
  dereferencing 135  
  disposing 112  
  lock count 135  
  locking 135  
  Mac OS 336, 383, 400  
  unlocking 147  
  versus pointers 85  
handles, Mac OS  
  locking and unlocking 344  
hasobjectcode 176, 220  
hasresources 176, 220  
hdlg resource 387  
help information 482  
  entry point 481  
helper application 218  
  launching 235  
Helper\_GetCompilerBrSymbols 209  
Helper\_GetCompilerBrSymbols Entry Point 270  
Helper\_Unmangle 210  
Helper\_Unmangle Entry Point 272  
hiding panel items 372

**I**

IDE  
  debugging setup 46  
  description 53  
  interaction with plug-ins 53  
  manipulating via Perl scripts 751  
  Master 46  
  settings panel API version variable 488  
  Slave 46  
  version 32  
IDE User Guide 38  
IDE version  
  determining 65  
IDL 735  
Import Project command 366  
importing settings data 366  
initializing  
  settings panel dialogs 354  
input files  
  effect on Perl scripts 751  
input focus  
  changing 373

managing 373  
**I**  
 Insertion Point class 716  
 installing  
     CodeWarrior 36  
     plug-in SDK 36  
     plug-ins 44  
         settings panel data 355  
 installing CodeWarrior 36  
 installing, Perl plug-ins 745  
 Interface Definition Language 735  
 interrupting long operations 148  
 invalidating  
     panel item 440  
 invalidating panel items 374  
 IR optimizer class 701  
 Is In Project 680  
 isGenerated 88, 104, 143, 227  
 isLaunchable 645  
 isPermanent 82, 107  
 ispermanent 231  
 isPostLinker 203, 299  
 isPreLinker 203, 299  
 isResourceFile 645  
 isresourcefile 176  
 item highlighting 373  
 item hit  
     determining 350  
 itemHit 396, 489  
 iterating access paths 114  
 ITypeLib 735

**J**

Java Compiler class 696  
 Java Linker 705  
 Java Project 686  
 Java Project class 686  
 JavaScript 28  
 JScript 28

**K**

kBadPrefVersion 352, 362, 519  
 kCandisassemble 200, 299  
 kCanimport 200, 299  
 kCanprecompile 200, 212, 300  
 kCanpreprocess 200, 212, 300

kCompAllowDupFileNames 200, 300  
 kCompAlwaysReload 201, 301  
 kCompEmitsOwnBrSymbols 201, 209, 271, 301  
 kCompMultiTargAware 201, 302  
 kCompRequiresFileListBuildStartedMsg 202, 302  
 kCompRequiresProjectBuildStartedMsg 201, 302  
 kCompRequiresSubProjectBuildStartedMsg 201,  
     303  
 kCompRequiresTargetBuildStartedMsg 201, 303  
 kCompRequiresTargetCompileStartedMsg 33  
 kCompUsesTargetStorage 201, 234, 257, 266, 304  
 kCurrentCompiledFile 112, 160, 189  
 kCurrentCWHelpInfoVersion 495  
 kDefaultLinkPosition 189  
 keyboards events, in Mac OS panels 389  
 kGeneratescode 200, 304  
 kGeneratesrsrcs 200, 304  
 kIgnored 207, 304  
 kInvalidCallbackErr 352, 520  
 kIsMPAware 201, 305  
 kIspascal 200, 305  
 kLaunchable 207, 305  
 kMissingPrefErr 352, 519  
 kPersistent 200, 305  
 kPrecompile 207, 306  
 kRsrcfile 207, 306  
 kSettingNotFoundErr 352, 519  
 kSettingOutOfRangeErr 353, 520  
 kSettingTypeMismatchErr 352, 520  
 kTargetGlobalPluginData 91, 189

**L**

latest API version  
     determining 124  
 LAttachemnt 386  
 launching  
     target executable 245  
 LBevelButton 386  
 LCaption 385  
 LCheckbox 385  
 LCheckBoxGroupBox 386  
 LEditText 385  
 limiting input text length 373  
 Line class 716  
 link errors  
     reporting 95

- 
- linkAgainstFile 288
  - linkage models 287
  - linkAllowDupFileNames 203, 306
  - linkAlwaysReload 204, 307
  - linker
    - CFM68K 704
    - combining debug information 223
    - getting output information 235
    - Java 705
    - Mac OS Merge 706
    - name unmangling 210
    - on-screen name 71
    - PowerPC 706
    - PowerPC PEF 707
    - storing output 234
    - target platform 208, 209
    - Win32 708
  - Linker classes 704
  - linker entry points 210
  - linker symbol unmangling 210
  - linkerGetTargetInfoThreadSafe 34, 80, 205, 219, 265, 310
  - linkers 214
    - request 213
  - linkMultiTargAware 203, 307
  - linkOutputDirectory 308
  - linkOutputFile 308
  - linkOutputNone 307
  - linkRequiresFileListBuildStartedMsg 204, 308
  - linkRequiresProjectBuildStartedMsg 204, 309
  - linkRequiresSubProjectBuildStartedMsg 204, 309
  - linkRequiresTargetBuildStartedMsg 204, 309
  - linkRequiresTargetLinkStartedMsg 204, 310
  - list box initialization
    - Windows 354
  - little-endian 360
  - loading
    - file text 122
  - loading files
    - headers and interfaces 110
    - non-project 88
    - project 88
  - loading settings data 92, 126
  - loading source text 213
  - localization
    - plug-in name 70
  - localized name 652
  - localized type names 271
  - location
    - of output directory 126
  - lock count
    - handles 135
  - low-level panel events
    - filtering, Mac OS 391
  - LPopupButton 386
  - LPopupGroupBox 386
  - LPushButton 385
  - LRadioButton 385
  - LRadioGroupView 386, 392
  - LSeparatorLine 386
  - LTextGroupBox 386
  - LView 385
- ## M
- Mac OS
    - handles 383, 400
    - plug-ins folder 45
  - Mac OS dialog manager 494
  - Mac OS errors
    - converting to IDE errors 136
  - Mac OS handles 336, 337
  - Mac OS Merge linker 61
  - Mac OS Merge linker class 706
  - Mach-O 687
  - Macintosh
    - plug-in binaries 58
  - magicCapLinker 204, 312
  - main () entry point
    - declaration 75
  - main entry point
    - result 149
    - settings panel 345
    - VCS 527
  - Main Plugin Entry Point 149
  - main() 110
  - main() function result 98
  - Make 681
  - Make command 213
  - Make command (IDE) 27

Make Project 670  
**m**alloc 81  
 managing input focus 373  
 manipulating  
   IDE via Perl scripts 751  
 Master IDE 46  
 Member Function class 719  
 memory  
   pointers versus handles 85  
 menu\_Clear 495  
 menu\_Copy 495  
 menu\_Cut 496  
 menu\_Paste 496  
 menu\_SelectAll 496  
 merging resources 234  
 message  
   displaying 105  
 Message Document class 728  
 message logging 51  
 messages  
   displaying 141  
 messagetypeError 190  
 messagetypeInfo 190  
 messagetypeWarning 190  
 Metrowerks  
   web site 40  
 Microsoft Scripting Technologies 732  
 Microsoft Windows scripting 731  
 Misc Classes 722  
 modification date 120  
   notifying IDE of change 144  
 modified file list  
   when determined 251  
 modified files  
   determining 251  
 modifying files 88  
 modifying panel settings 343  
 mouse events, in Mac OS panels 389  
 moveHi 135  
 MW\_Perl 746  
 MW\_Perl.dll 746  
 MWPerl\_Panel.dll 746

**N**

naming conventions  
 panel API 400

Negotiation Mode  
 VCS 554  
 negotiation request 532  
 new features 30  
 newestAPIVersion 68, 341, 484  
 notifying  
   IDE of file changes 144  
 numeric values  
   getting 431

**O**

Object Classes 719  
 object code  
   storing 282  
   storing externally 228  
 object code, PPC  
   alternate entry point hunks 603  
   array type 618  
   cross-reference hunks 604  
   data header 610  
   data hunks 601  
   data section 597  
   enumerated type 619  
   function data section 611  
   header 594  
   name table section 625  
   Object Pascal hunks 608  
   Pascal array type 621  
   Pascal enumerated type 623  
   Pascal set type 623  
   Pascal string type 624  
   Pascal subrange type 622  
   PEF import hunks 605  
   pointer type 617  
   regular hunks 599  
   reserved hunks 610  
   simple hunks 598  
   source file specification hunks 607  
   structure 594  
   structured type 618  
   type data section 614

object data  
 format of 214, 263  
 freeing 242  
 loading 263  
 retrieving 228  
 storing 227, 255, 268

object files

- 
- determining location of 228
  - recommended location 255
  - storing externally 228
  - object files, external**
    - determining location 254
  - obsolete routines 97
  - obtaining version 117
  - oldtargfile 365, 490
  - OLE/COM Object Viewer 733
  - Open 661
  - Open Browser 683
  - OPENFILENAME 421
  - operating system codes 208
  - operating system error
    - obtaining 117
  - operating systems
    - target 270, 651
  - optimization
    - PowerPC 699
  - optimizer 701
  - originalPrefs 344, 488
  - OS error codes 97
  - output directory 89
    - determining location 126
    - plug-in 47
  - output files
    - effect on Perl scripts 751
  - overlay
    - getting information 90
  - overlay file
    - getting information 90
    - getting position 127
  - overlay groups 86
    - counting 90, 128
    - getting information 90, 128
  - overlays 86
    - getting information 129
- P**
- panel codes 74
  - panel families
    - standard 74
  - panel family 73
  - panel information 484
  - panel items
    - activating 418
    - appending 419
- cancelling redraw 451
  - creating 419
  - determining item hit 425
  - enabling and disabling 423
  - framing 453
  - getting control from 455
  - getting item rectangle 457
  - getting port 458
  - redrawing 440
  - showing 450
  - panel settings**
    - current 424
    - getting 356
    - putting 355
  - panel. See **settings panel**
  - panelfamily 341, 484
  - PanelFlags 68, 339, 483
  - PanelParamBlkPtr 485
  - PanelParameterBlock 332, 334, 485
  - panelscope 341, 484
  - panelScopeGlobal 497
  - panelScopeProject 497
  - panelScopeTarget 498
  - Pascal
    - string limitations on Mac OS 372
  - Pascal Compiler class 697
  - Pascal Warnings class 698
  - Path Information class 688
  - path reset flag
    - setting 449
  - Perl 28
    - avoiding link errors in scripts 750
    - effects of input and output files 751
    - manipulating the IDE 751
    - special scripting considerations 750–752
    - StdIn usage in scripts 750
  - Perl Panel 746
  - Perl plug-ins
    - installation on Mac OS 746
    - installation on Windows 746
  - Perl script example (Mac OS) 749
  - Perl Scripting 745
  - Perl scripting 745
    - with IDE plug-in API 749–750
  - Perl software plug-ins
    - installation 745–747
  - Perl target settings panel 748

---

configuring 747–748  
Perl, installing plug-ins 745  
PerlDLL.dll 746  
permanent memory allocation 107  
persistent plug-in data 91  
platform  
  supported 208, 209  
pluggin SDK  
  setup 41  
plug-in  
  binary formats 58  
  capabilities, reporting 150  
  common requests 78  
  compiler 54  
  definition 26  
  editor 56  
  enabling 270  
  entry point calling conventions 58  
  installation 44  
  interaction with IDE 53  
  linker 54  
  loading and unloading 231  
  nontraditional applications 54  
  persistent data 91  
  post-linker 54  
  pre-linker 54  
  settings panel, definition 55  
  types 54  
  VCS, definition 55  
plug-in API 117  
plug-in context 99, 332, 333, 400  
plug-in data 91  
  external 94  
  loading 130  
  retrieving 92  
  storing 91  
  storing with a file 91  
  storing with a target 91  
  type code 92  
plug-in debugging 46  
  breakpoints ignored 52  
  diagnostics error level 48  
  message logging 51  
  options 48  
plug-in diagnostics 48  
plugin installation  
  automatic 45  
plug-in requests  
  reentrancy 80  
plugin requests  
  common 78  
plug-in SDK  
  contents 40  
  debugging setup 46  
  installation steps 41  
  installing 36  
  introduction 39  
  troubleshooting 51  
plug-in type code  
  preference panel 69  
plug-in type codes 67  
plug-in unloading 225  
plug-ins  
  installing 44  
  installing on Mac OS 45  
  installing on Windows 44, 45  
  Perl, installing 745  
  recognizing 45  
  types 53  
plugins  
  types 26  
pointer memory 81  
  allocating 106  
pointers  
  disposing 113  
  freeing 113  
  stale 147  
  vesus handles 85  
popup menu 432  
  setting value 445  
post-linker 709  
  storing output 235  
Post-linkers 214  
PowerPC assembler 695  
PowerPC code generation 700  
PowerPC Disassembler class 703  
PowerPC Linker class 706  
PowerPC optimization 699  
PowerPC PEF linker class 707  
PowerPC Project 684  
PowerPlant controls  
  restrictions 386  
PPC CodeGen class 700  
PPC Mach-O Project 687  
PPC Project class 684

---

- 
- PPCAsm 695
  - Precompile 671
  - Precompile command 212, 261
  - precompiled header 240
    - caching 240
    - save location 252
  - precompiled header caching
    - determining if enabled 230, 260
    - verifying enabled 241
  - precompiled headers 212
    - caching 230
  - precompiling 212
    - automatic 259
    - determining if enabled 261
  - preference change notification 524
  - preferences. See settings panel.
  - prefixes
    - settings panel routine names 337
  - prefsDesc 397, 490
  - prefsKeyword 397, 490
  - Pre-linkers 214
  - Preprocess 671
  - Preprocess command 212
  - preprocessing 212
    - determining if enabled 262
  - Print 662
  - printf statements 51
  - processor codes 208
  - processors
    - target 651
  - progress of
    - target 33
  - progress window 241
  - progress, displaying 95
  - project
    - counting files in 132
    - file specification to 488
  - Project Classes 684
  - project components 86
    - specifying 87
  - Project Document class 728
  - project file
    - getting file specification 132
  - Project File class 693
  - project files
    - adding 88
    - counting 132
  - project groups 86
  - project information 89
  - Project Inspector class 728
  - project structure 86
- Q**
- QuickStart 36
  - Quit 662
- R**
- radio button 432
    - setting value 445
  - radio buttons, handling item hits on Mac OS 392
  - recompilation, forcing 361
  - recompile 490
  - recompile flag 361
    - setting 447
  - recursive request 532
  - redrawing
    - panel item 440
  - redrawing panel items 374
  - reentrancy 219
    - during request handling 80
  - registry 288
  - relative path
    - choosing 421
    - converting to human-readable string 437
    - creating Apple Event from 466
    - extracting from Apple Event 461
    - resolving to full path 143
  - relative paths 90
  - release notes 30
  - relink 490
  - relink flag 361
    - setting 447
  - relinking, forcing 361
  - relocation
    - handles 83
  - Remove Binaries 672
  - Remove Files 665
  - Remove Object Code 672
  - removing
    - combo box items 422
  - reparse 490
  - reparse flag 361
    - setting 448

---

reporting compile and link errors 95  
reporting errors to IDE 98  
reporting OS error codes 98  
reqAbout 191, 559  
reqActivateItem 393, 498  
reqAEGetPref 397, 499  
reqAESetPref 397, 499  
reqByteSwapData 500, 517  
reqCheckSyntax 213, 312  
reqCompDisassemble 219, 313  
reqCompile 259, 313  
reqDatabaseConnect 558  
reqDatabaseDisconnect 558  
reqDatabaseVariables 559  
reqDeactivateItem 393, 500  
reqDisassemble 219, 314  
reqDragDrop 396, 501  
reqDragEnter 395, 502  
reqDragExit 396, 503  
reqDragWithin 396, 504  
reqDrawCustomItem 392, 393, 504  
reqFileAdd 560  
reqFileBranch 561  
reqFileCheckin 561  
reqFileCheckout 561  
reqFileComment 561  
reqFileDelete 562  
reqFileDestroy 562  
reqFileDifference 562  
reqFileGet 562  
reqFileHistory 563  
reqFileLabel 563  
reqFileListBuildEnded 224, 315  
reqFileListBuildStarted 224, 315  
reqFileProperties 563  
reqFilePurge 563  
reqFileRename 564  
reqFileRollback 564  
reqFileShare 564  
reqFileStatus 564  
reqFileUndoCheckout 565  
reqFileVersion 565  
reqFileView 565  
reqFilter 391, 505  
reqFindStatus 393, 394, 505  
reqFirstLoad 355, 506  
reqGetData 92, 356, 506  
reqGetData Request 343  
reqGetFactory 358, 507  
reqGetFactory Request 343  
reqHandleClick 389, 508  
reqHandleKey 389, 508  
reqIdle 191, 538, 559  
reqInitialize 557  
reqInitDialog 419, 446, 509  
reqInitialize 192, 225, 353  
reqInitialize request 78  
reqInitPanel 353, 509  
reqItemHit 366, 372, 391, 425, 427, 443, 510  
reqLink 214, 315  
reqObeyCommand 393, 394, 511  
reqPrefsChange 192, 538, 560  
reqPreRun 316  
reqProjectBuildEnded 224, 316  
reqProjectBuildStarted 224, 317  
reqPutData 355, 443, 512  
reqReadSettings 35, 408, 416, 512  
reqRenameProject 365, 513  
reqSetupDebug 364, 513  
reqSubProjectBuildEnded 224, 317  
reqSubProjectBuildStarted 224, 318  
reqTargetBuildEnded 224, 318  
reqTargetBuildStarted 224, 318  
reqTargetInfo 34, 80, 216, 219, 234, 256, 265, 319  
reqTargetLinkEnded 223, 320  
reqTargetLinkStarted 223, 320  
reqTargetLoaded 82, 107, 210, 231, 235, 266, 320  
reqTargetPrefsChanged 321  
reqTargetUnloaded 82, 107, 231, 235, 266, 321  
reqTermDialog 355, 514  
reqTerminate 193, 225, 353, 557  
reqTerminate request 79  
reqTermPanel 353, 514  
request 488  
    compiler 211  
    compiling 211  
    determining 131, 149  
    linker 211  
    precompiling 212  
    preprocessing 212  
    target information 216

---

request handling 75  
requests  
  compiler and linker, start and end 223  
  determining, settings panel 345  
require directive 747  
requirements  
  overview 35  
  programming experience 36  
reqUpdatePref 344, 362, 515  
reqValidate 361, 516  
reqWriteSettings 35, 409, 417, 517  
rescan flag 361  
rescanning for project files 361, 449  
reset 489  
Reset File Paths 681  
reset paths flag  
  setting 449  
resizing  
  handle 85  
resizing handles 85  
resolving a relative path 143  
Resorcerer 730  
Resorcerer™ 384  
resource attributes 133  
resource data  
  retrieving 228  
  storing 227, 282  
resource disassembly 221  
resource file  
  saving 264  
resource files  
  selecting 253  
resources  
  instead of entry points 61  
  specifying plug-in capabilities 61  
result  
  main() 98  
retrieving plug-in data 92  
returning errors 109  
revert flag  
  setting 366, 450  
Rez 384, 698  
Rez Resource Compiler class 698  
Run 662, 672  
Run command 218  
run helper 218, 235, 287  
  executing 288

Run Project 672  
runHelperIsRegKey 288  
runHelperName 288  
runtime environment  
  setup for helper application 235  
runtime information  
  obtaining 64  
runtime options 218  
Runtime Settings panel 34

**S**

sample projects  
  building 44  
  output directory 43  
  test project 44  
Save Error Window As 666  
scripting  
  resources 387  
  using Perl 745  
scripting the IDE 27  
Segment class 692  
segment information  
  getting 90, 133  
segments 86  
Select 666  
selecting files 253  
Selection-Object class 717  
Set 675  
Set Current Target 679  
Set Default Project 679  
Set Modification Date 666  
Set Preferences 677  
Set Project File 678  
Set Segment 680  
settings panel  
  differences between platforms 332  
setting data  
  scheme for reading 370  
setting factory and revert flags 366  
setting panel families 650  
setting up debugging 364  
settings  
  XML 483  
settings data 364  
  current 424  
  factory defaults 427

---

loading 92, 126  
loading for a different panel 436  
original 434  
restoring factory values 358  
schema for updating 362  
storing 92  
structured XML elements 369  
supported XML types 367  
updating 362  
settings handles  
  accessing 343  
settings panel  
  data structures 481  
  data version 344  
  definition 338  
  determining request 345  
  dropin type 340  
  entry points 339  
  main entry point, Mac OS 333  
  main entry point, Windows 333  
  parameter block, direct access on Mac OS 334  
  request variable 488  
  state accessor calls 333  
settings panel API version 488  
settings panel context 333  
settings panel data  
  updating 344  
settings panel family 341  
settings panel requests  
  closing a panel dialog 355  
  extracting control values 356  
  getting data 356  
  getting factory settings 358  
  initializing a panel dialog 354  
  installing control values 355  
  loading a new target 355  
  putting data 355  
  reading XML data 370  
  renaming a project 365  
  responding to an item hit 365  
  setting up debugging 364  
  swapping byte ordering 359  
  updating settings 362  
  user interaction 365  
  validating settings 361  
Windows 377  
  writing XML data 370  
settings panels  
  'DITL' versus 'PPob' 384  
activating items 418  
adding Balloon Help 387  
appending items 419  
assigning control IDs 380  
background, Mac OS 386  
base items 433  
base number of items 350  
cancelling item redraw 451  
checkboxes, Mac OS 391  
closing 355  
constructing user interface, Windows 378  
control restrictions, Mac OS 386  
data format 344  
determining item hit 350, 425  
determining panel request, Mac OS 389  
dialog initialization 354  
Edit menu item handling 393  
enabling and disabling items 372  
error code 351  
extracting control values 356  
factory default settings 427  
factory defaults 358  
framing items 453  
getting control from item 455  
getting item rectangle 457  
getting port 458  
grouping 650  
handling low-level events, Mac OS 389  
hiding and showing items 372  
initialization 353  
installing control values 355  
invalidating items 374  
layout, Mac OS 385  
low-level events, Mac OS 391  
main entry point 345  
modifying data 343  
Pascal versus C strings 337  
PowerPlant controls 385  
radio button items, Mac OS 392  
redrawing items 374, 440  
resource creation, Windows 378  
resource list 652  
resources, Mac OS 384  
responding to user interaction 366  
returning results to IDE, Mac OS 389  
setting text items 443  
showing items 450  
strict API 342  
supported control types, Mac OS 385

---

- 
- supported control types, Windows 381
  - termination 353
  - validating item content 373
  - SFTypeList 422
  - showing panel items 372
  - Single Class Browser class 728
  - Single Class Hierarchy class 728
  - sizes
    - of object code and data 283
  - Slave IDE 46
  - source-to-source compilation 226
  - specifying debugging host 47
  - specifying dependencies 111
  - specifying project components 87
  - specifying target platforms 208
  - stale pointers 83, 147
  - standard 74
  - start
    - build 33
  - static text 432
    - getting 429
    - getting and setting 371
    - setting 443
    - setting to a handle 444
    - setting value 445
  - status
    - displaying 95
  - StdIn usage
    - in Perl scripts 750
  - storage 488
  - storing linker information 234
  - storing object data 268
  - storing plug-in data 91
  - storing settings data 92
  - strict panel API, Mac OS 342
  - subprojects 86
  - supportsByteSwapping 517
  - supportsTextSettings 370, 518
  - suppressload 110
  - Symbol Browser class 728
  - symbol file format
    - Mac OS 223
  - symbol type names
    - entry point 642
  - symbol unmangling 257, 291
    - entry point 642
  - symbolic information
    - precompiled 212
    - storing in external files 223
  - symbolics file
    - location 287
  - symName 271
  - symUIName 271
  - Syntax Coloring class 718
- T**
- target
    - getting output directory 89
    - providing information 216
  - Target class 724
  - target data directory
    - determining 94
  - target executable
    - launching 245
  - Target File class 724
  - target files
    - getting location 89
  - target info
    - modifying (by post-linker) 217
  - target information
    - getting 89, 256
    - reporting to IDE 216
    - setting 265
  - target name
    - getting 89, 134
  - target operating systems 651
  - target platforms
    - specifying 208
  - target processors 651
  - Target Settings class 689
  - target storage 256
    - and unmangling 273
    - disposing early 234
    - lifetime of 266
    - managing 230, 257, 266
  - targetCPU68K 322
  - targetCPUAny 323
  - targetCPUEmbeddedPowerPC 323
  - targetCPUi80x86 323
  - targetCPUMips 324
  - targetCPUNECv800 324
  - targetCPUPowerPC 325
  - targetfile 488
  - Targeting Mac OS 38

Targeting Win32 38  
 targetOSAny 325  
 targetOSEmbeddedABI 326  
 targetOSMacintosh 326  
 targetOSMagicCap 326  
 targetOSOS9 327  
 targetOSWindows 327  
 targets 86  
 targetStorage 210  
 temporary files 89, 218  
     storing 133  
 terminology resource, AppleScript 387  
 test project 44  
 text  
     displaying in a window 95  
     disposing 139  
     freeing 139  
     getting 430  
     getting and setting 371  
     validation 429  
 Text class 717  
 Text Document class 725  
 text files  
     loading 122, 213  
 text length  
     limiting 373  
 third-party editor support 56  
 thread synchronization 219  
 threading issues 219, 265  
 thread-safety 34  
 tm structure 35  
 toEndian 491, 500, 506  
 toendian 360  
 ToolServer Worksheet class 728  
 top-level XML settings 367  
 Touch 673  
 touched files  
     determining 251  
 tracking dependencies 229  
 troubleshooting 51  
 type code 157  
 typedefs  
     platform dependent 62  
 types of plug-ins 53, 54  
 types of plugins 26

**U**  
 unloading plug-ins 225, 231  
 unlocking a handle 147  
 unmangling 210, 266, 273, 291  
     entry point 642  
     storing associated data 257  
     storing information for 210  
 Update Project 674  
 updating panel data 344, 362  
 URL syntax 288  
 user break 96, 148  
 user input  
     validating 373  
 usesStrictAPI 488, 494, 518  
 useTempMemory 108  
 validating input 373  
 VBScript 28  
 VCS 523  
     capability flags 536  
     command support negotiation 527  
     command variants 532  
     compatibility checking 528  
     entry points 535  
     file lists 534  
     file processing 527  
     initialization and termination 526  
     main entry point 527  
     negotiation request handling 530  
     notifications 527  
     phases 525  
     request handling 530  
     returning from request 528  
 VCS API  
     version 7 changes 523  
     version 8 changes 523  
     version 9 changes 523  
 vcsDoesntPrePostRecursive 536, 537  
 vcsDoesntPrePostSingleFile 536  
 vcsRequiresEvents 536, 537  
 vcsSupportsPrefsChanged 536, 538  
 vcsWantsIdle 536, 538  
 version 488  
     of settings panel data 344  
 version control system 523  
 Version Control System Setup 725  
 version numbers 31  
 version of plug-in API

- 
- determining 117
  - determining latest 124
  - visual difference 524
  - Visual Studio 6.0 734
  - WaitNextEvent 396
  - warnings
    - displaying 141
    - reporting 95
  - web site
    - Metrowerks 40
  - wildcard 651
    - CPU and OS code 208
  - wildcard, DOS 421
  - Win32 code generation 702
  - Win32 linker 708
  - Win32 project class 686
  - Win32/x86 CodeGen class 702
  - Win32/x86 Linker class 708
  - Win32/x86 Project 686
  - window
    - creating 109
  - Window class 718
  - Windows
    - plugin binaries 58
    - plug-ins folder 44, 45
  - windows
    - operating system, displaying 96, 137, 138
  - Windows Resource Compiler 699
  - Windows scripting 731
  - Windows settings panel resources 378
  - Windows, settings panels
    - assigning control IDs 380
    - layout requirements 381
    - supported control types 381
  - WinHelp 481, 482
  - WinRC 699
  - working directory 218, 236
    - getting 258
  - x86 code generation 702
  - x86 linker 708
  - x86 project class 686
  - XCOFF 593
  - XML
    - array and structure elements 369
    - arrays, number of elements 410
    - importing and exporting 35, 342, 366
    - reading an array or structure setting 413
  - reading array elements 408
  - reading nested structures and arrays 409
  - reading structure fields 416
  - reading structured boolean values 411
  - reading structured floating point values 412
  - reading structured integer values 413
  - reading structured relative path values 415
  - reading structured string values 415
  - reading top-level boolean values 468
  - reading top-level floating point values 468
  - reading top-level integer values 469
  - reading top-level relative path values 470
  - reading top-level string values 471
  - setting data import scheme 370
  - setting ID 483
  - setting IDs 368
  - setting names 367
  - setting types 367
  - structured types 368
  - top-level settings 367
  - writing nested structures and arrays 409, 417
  - writing structured boolean values 472
  - writing structured floating point values 473
  - writing structured integer values 474
  - writing structured relative path values 475
  - writing structured string values 476
  - writing top-level boolean values 477
  - writing top-level floating point values 478
  - writing top-level integer values 478
  - writing top-level relative path values 479
  - writing top-level string values 480