



User Guide and Reference Manual

Version 3.0
September 2012



Scientific Toolworks, Inc.
249 East Tabernacle, Suite 200
St. George, UT 84770

Copyright © 2012 Scientific Toolworks, Inc. All rights reserved.

The information in this document is subject to change without notice. Scientific Toolworks, Inc., makes no warranty of any kind regarding this material and assumes no responsibility for any errors that may appear in this document.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFAR 252.227-7013 (48 CFR). Contractor/Manufacturer is Scientific Toolworks, Inc., 249 East Tabernacle, Suite 200, St. George, UT 84770.

NOTICE: Notwithstanding any other lease or license agreement that may pertain to or accompany the delivery of this restricted computer software, the rights of the Government regarding use, reproduction, and disclosure are as set forth in subparagraph (c)(1) and (2) of Commercial Computer Software-Restricted Rights clause at FAR 52.227-19.

Part Number: USTAND3.0-GEN-UG-629 (9/12)

Contents

Chapter 1	Introduction	
	What is Understand?	13
	Licensing Issues	14
	Languages Supported	15
	For Those Who Don't Like to Read Manuals	16
Chapter 2	Parts and Terminology	
	Using Understand Windows	18
	Understand Terminology	19
	Parts.	20
	Starting Understand	21
	Other Ways to Run Understand	22
	Context Menus Are Everywhere	23
	Quickly Find Things in Your Source	25
	Entity Filter	25
	Entity Locator	26
	Instant Search	26
	Find in Files	27
	Favorites	27
	Information Browser	28
	Source Editor	29
	Architecture Browser	30
	Graphical Views	31
	ASCII and HTML Reports	32
	APIs for Custom Reporting	32
Chapter 3	Configuring Your Project	
	About Understand Projects	34
	The Understand Project Database	34
	Creating a New Project	35
	New Project Wizard	35
	Project Configuration Dialog	39
	Languages Category	41
	Files Category	42
	Adding Directories	43
	Adding Files	44
	Removing Directories and Files	45
	Setting Overrides	45
	Scanning Watched Directories	46
	Setting File Portability	47
	File Types	48
	File Options	49

Scheduled Activities	50
Metrics	52
Metrics > Selected Category	53
Reports	54
Reports > Output Category	54
Reports > Options Category	55
Reports > Selected Category	56
Visual Studio	57
Annotations	59
Ada Options	61
Ada > Macros Category	63
Assembly Options	65
COBOL Options	67
COBOL > Copybooks Category	67
C++ Options	69
C++ > Includes Category	71
C++ > Includes > Auto Category	72
C++ > Includes > Ignore Category	72
C++ > Includes > Replacement Text	73
C++ > Macros Category	73
C++ > Macros > Undefines Category	75
C++ (Strict) Options	76
C++ (Strict) > Includes Category	77
C++ (Strict) > Includes > Frameworks Category	78
C++ (Strict) > Includes > Prefix Headers Category	79
C++ (Strict) > Macros Category	79
C# Options	79
FORTRAN Options	80
Fortran>Includes Category	82
Other Fortran Categories	82
Java Options	83
Java > Class Paths Category	83
JOVIAL Options	85
Jovial > !Copy Category	86
Pascal Options	87
Pascal > Macros Category	88
Pascal > Namespaces Category	88
Pascal > Standard Library Paths Category	88
Pascal > Search Paths Category	89
PL/M Options	89
PL/M>Includes Category	90
Python Options	91
Python > Imports Category	91
VHDL Options	91
Web Options	92
Setting General Preferences	93
General Category	94

User Interface Category	96
User Interface > Lists Category	97
User Interface > Alerts Category	98
User Interface > Windows Category	99
Key Bindings Category.	100
Analyze Category.	101
Configure Category	102
Command Window Category	102
Portability Category	103
Dependency Category	104
Editor Category	105
Editor > Advanced Category	107
Editor > Styles Category	110
Editor > Navigation Category	111
Editor > External Editor Category.	112
Graphs Category	113
Analyzing the Code.	114
Using the Missing Header Files Tool	116
Using the Undefined Macros Tool	118
Converting an Understand 1.4 Project	119

Chapter 4

Exploring Your Codebase

PLEASE RIGHT CLICK	121
Various Windows Explained...	122
Entity Filter	123
Using the Filter Field	124
Customizing the Display.	124
Root Filters.	124
Information Browser	125
Drilling Down A Relationship	126
Displaying More or Less Information	126
Searching the Information Browser	127
Syncing the Information Browser	127
Visiting Source Code	128
Visiting References	128
Viewing Metrics	129
Saving and Printing Information Browser Text.	129
Entity History	130
Project Browser	130
Exploring a Hierarchy	132
Dependency Browser	133
Favorites.	135
Creating a Favorite Entity.	135
Creating a Favorite View	136
Using a Favorites Group	136
Creating a Plain Text Favorite	138

Chapter 5

Searching Your Source

Searching: An Overview	140
Instant Search	141
Find in Files	143
Find Results	145
Replace in Files	146
Entity Locator	148
Resizing Columns	148
Long versus Short Names	148
Column Headers	149
Choosing Columns	149
Filtering the List	149
Finding Windows	152
Source Visiting History	153
View Menu Commands	153
Displaying Toolbars	154
Searching in a File	155
Find Next and Previous	155
Find & Replace	155
Contextual Information Sidebar	156

Chapter 6

Editing Your Source

Source Editor	158
Scope List	159
Status Icons	159
Status Line	159
Selecting and Copying Text	160
Browse Mode	160
Context Menu	161
Hover Menu	161
Saving Source Code	162
Other Editing Features	163
Bracket Matching	163
Folding and Hiding	163
Splitting the Editor Window	164
Commenting and Uncommenting	164
Fixing Indentation	164
Changing Case	164
Line Wrapping	165
Insert and Overtyping Modes	165
Sorting Lines Alphabetically	165
Keyboard Commands	165
Recording and Replaying Macros	165
Creating and Opening Files	165
Bookmarking	166
Managing Source Editor Tabs	167
Changing the Source Code Font Size	167
Annotations	168

Adding an Annotation	168
Editing an Annotation	170
Deleting an Annotation	170
Managing Annotation Files and Display	170
Searching Annotations	171
Filtering Annotations	172
Managing Orphaned Annotations	173
Printing Source Views	174

Chapter 7

Architecting Your Codebase

About Architectures	176
Using the Architecture Browser	177
Exploring Architectures	177
Viewing Architecture Dependency Graphs	179
Graph Customizer Toolbar	180
Graph Customizer Fields	180
Viewing Architecture Metrics	183
Managing Architectures	184
Creating an Architecture	185
Using the Architecture Wizard	186
Building an Architecture	187
Using XML to Manage Architectures	189
Exporting Architectures to XML	189
Importing XML Architectures	189

Chapter 8

Using Reports

Configuring Reports	191
Customizing Report Colors	193
Generating Reports	193
Viewing Reports	194
An Overview of Report Categories	195
Augment with the PERL or C API	196
Cross-Reference Reports	197
Data Dictionary Report	197
Program Unit Cross-Reference Report	198
File Contents Report	198
Object Cross-Reference Report	199
Type Cross-Reference Report	199
Class and Interface Cross-Reference	199
Macro Cross-Reference	200
Include File Cross-Reference	200
Exception Cross-Reference Report	201
Structure Reports	202
Declaration Tree	202
Class Extend Tree	203
Invocation Tree Report	203
Simple Invocation Tree Report	203
With Tree Report	204

Simple With Tree Report	204
Generic Instantiation Report	204
Renames Report	204
Import Report	204
Quality Reports	205
Program Unit Complexity Report	205
FORTRAN Extension Usage Report	206
Implicitly Declared Objects Report	207
Uninitialized Items	207
Unused Variables and Parameters	207
Unused Objects Report	207
Unused Types Report	207
Unused Program Units Report	208
Uses Not Needed Report	208
Withs Not Needed Report	208
Metrics Reports	209
Project Metrics Report	209
Class Metrics Report	210
Class OO Metrics Report	210
Program Unit Metrics Report	210
File Metrics Report	211
File Average Metrics Report	212

Chapter 9

Using Metrics

About Metrics	214
Metrics Summary	215
Metrics Browser	216
Exporting Metrics to HTML	217
Exporting Metrics to a CSV File	218
Configuring Metric Charts	220
Using the Metrics Treemap	223
Exporting Dependency Metrics	225
Exporting Dependencies to a CSV File	226
Exporting Dependencies to a CSV Matrix File	227
Exporting Dependencies to Cytoscape	227

Chapter 10

Using Graphical Views

Project Overview Graphics	229
Graphical View Browsers	231
Hierarchy Views	232
Structure Views	233
General Commands for Using Graphical Browsers	233
Filtering Out Entities	235
Reuse Checkbox	236
Sync Checkbox	236
Graph Options	236
Types of Views	237
Hierarchy View Types	237

Hierarchy View Examples	238
Structure View Types	240
Structure View Examples	241
Graphical Notation	245
Controlling Graphical View Layout	245
Called by Menu	246
Constants Menu	246
Default Members Menu	246
Dependent Of Menu	246
Dependent Menu	246
Depth	246
Duplicate Subtrees Menu	246
Expand Recursive Notes	247
Expand Repeated Notes	247
Extended By Menu	247
Extends Menu	247
External Functions Menu	247
Filename Menu	247
Function Pointer Menu	247
Globals Menu	247
Implements Menu	247
Implemented By Menu	248
Imports Menu	248
Included By Menu	248
Includes Menu	248
Inherits Menu	248
Inherited By Menu	248
Intrinsic Menu	248
Invocations Menu	249
Layout Menu	249
Level Menu	250
Locals Menu	250
Members Menu	250
Name Menu	250
Objects Menu	251
Operators Menu	251
Parameters Menu	251
Private Members Menu	251
Protected Members Menu	251
Public Members Menu	251
Renames Menu	251
Routines Menu	251
Scale Menu	252
Sort Menu	252
Spacing Menu	252
Sql Menu	252
Static Menu	252
Text Menu	252
Types Menu	253
Typetext Menu	253

	Unknown Menu	253
	Unresolved Menu	253
	Usedby Menu	253
	Uses Menu	253
	Variables Menu	253
	Withs Menu	253
	With Bys Menu	253
	Controlling Cluster Graph Layout	254
	Saving Graphical Views	255
	Saving Views to Files	255
	Saving Views as Visio Files	256
	Saving Views as DOT Files	256
	Printing Graphical Views	257
	Graphical View Printing	257
Chapter 11	Using CodeCheck for Standards Verification	
	About CodeCheck	260
	Running a CodeCheck	261
	Files Tab	261
	Checks Tab	262
	Viewing CodeCheck Results	264
	Using the Result Log	264
	Using the Results by File Tab	265
	Using the Results by Check Tab	266
	Using the Result Locator	267
	Using the Result Treemap	268
	Printing and Exporting Results	269
	Ignoring Checks and Violations	270
	Using CodeCheck Configurations	271
	Writing CodeCheck Scripts	272
	Installing Custom Scripts	273
Chapter 12	Comparing Source Code	
	Comparing Files and Folders	275
	Comparing Entities	277
	Comparing Text	278
	Exploring Differences	279
	Code Comparison	279
	Patch File	282
	Difference List	282
Chapter 13	Running Tools and External Commands	
	Configuring Tools	284
	Variables	286
	Adding Tools to the Context Menus	291
	Adding Tools to the Tools Menu	292
	Adding Tools to the Toolbar	293

Importing and Exporting Tool Commands	294
Running External Commands	295

Chapter 14

Command Line Processing

Using the und Command Line	298
Getting Help on Und.	300
Creating a New Project	300
Adding Files to a Project	300
Removing Items from a Project	301
Getting Information about a Project	301
Modifying Project Settings	301
Importing into a Project	302
Exporting from a Project.	302
Analyzing a Project	303
Generating Reports	303
Generating Metrics	303
Using CodeCheck	304
Running Perl Scripts	304
Creating a List of Files	304
Using the understand Command Line	305
Using Buildspy to Build Understand Projects	306

Chapter 15

Quick Reference

File Menu	308
Edit Menu	309
Search Menu	309
View Menu	310
Project Menu	310
Reports Menu	311
Metrics Menu	311
Graphs Menu	312
CodeCheck Menu	312
Annotations Menu	312
Tools Menu	313
Window Menu	314
Help Menu	314

Chapter 1

Introduction

This chapter introduces the *Understand* software.

This manual assumes a moderate understanding of the programming language in which your project is written.

This chapter contains the following sections:

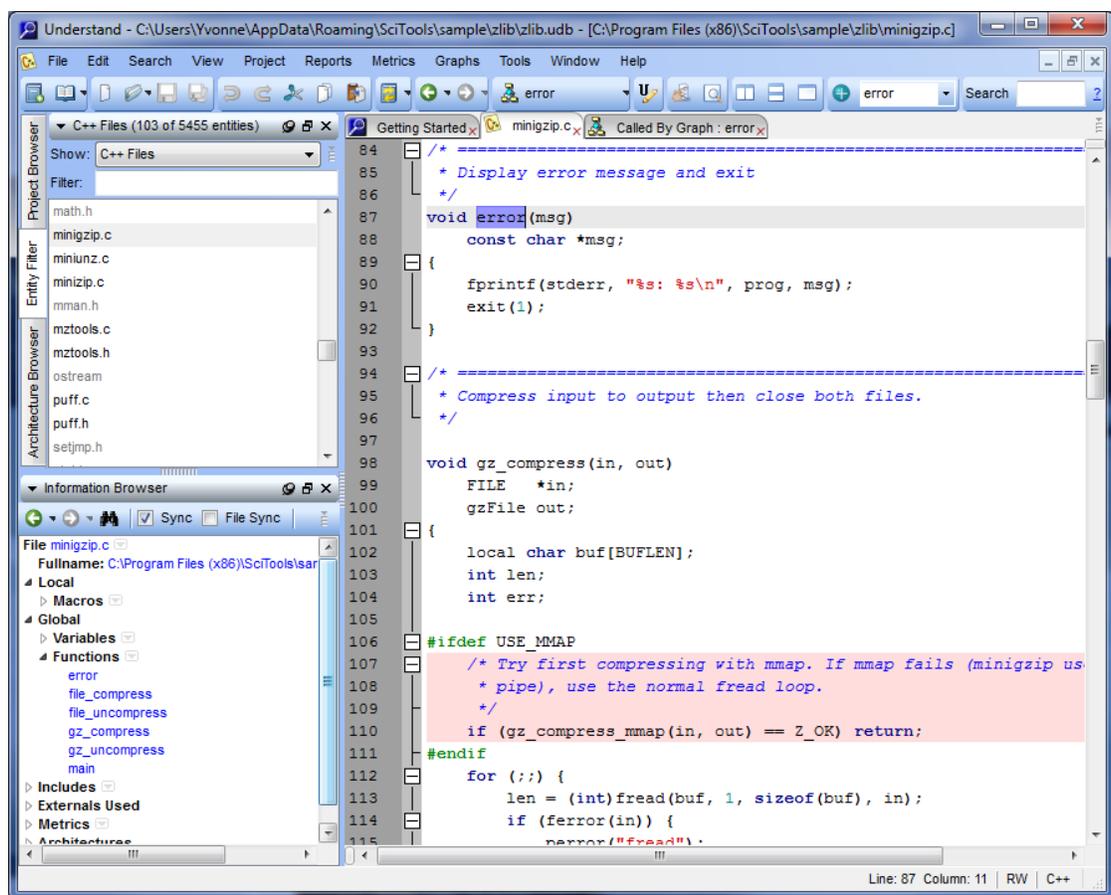
Section	Page
What is Understand?	13
Licensing Issues	14
Languages Supported	15
For Those Who Don't Like to Read Manuals	16

What is Understand?

Understand is a source code analysis and metrics tool. It is designed to help maintain and understand large amounts of legacy or newly created source code. It provides a cross-platform, multi-language, maintenance-oriented IDE (interactive development environment).

The source code analyzed may include C, C++, C#, Objective C/Objective C++, Ada, Java, Pascal/Delphi, COBOL, JOVIAL, VHDL, FORTRAN, PL/M, Python, PHP, HTML, CSS, JavaScript, and XML.

It offers code navigation using a detailed cross-referencing, a syntax-colorizing “smart” editor, and a variety of graphical reverse engineering views.



Understand creates a repository of the relations and structures contained within the software project. The repository is then used to learn about the source code.

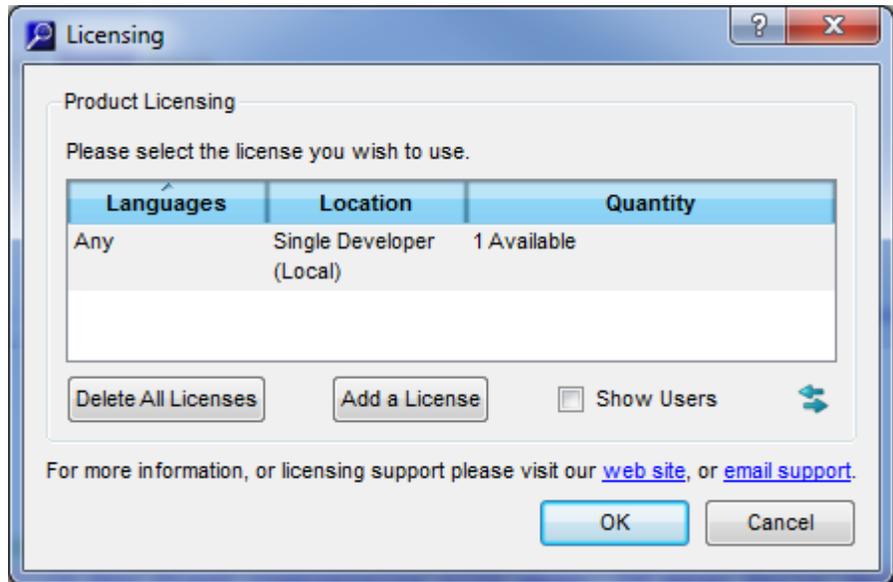
Understand has analysis features that help you quickly answer questions such as:

- What is this entity?
- Where is it changed?
- Where is it referenced?
- Who depends on it?
- What does it depend on?

Understand has architecture features that help you create hierarchical aggregations of source code units. You can name these units and manipulate them in various ways to create interesting hierarchies for analysis.

Licensing Issues

To view or change the license being used, choose **Help->Licensing** from the menus, select the license you want to use, and then restart *Understand*.



If you have multiple licenses, you can select the one you want to use here. If you have a new license key, click **Add a License** and choose to add either an evaluation or Single Developer License (SDL) or the name of a license server.

If you are using a floating license, you can check the **Show Users** box to see the currently active users. Click the double-arrow icon to refresh the license use information.

Languages Supported

The following list provides a brief overview of the language versions and/or compilers that *Understand* supports:

- **Ada:** *Understand* supports Ada83, Ada95, and Ada05 code, separately, or in combination.
- **Assembly:** Assembly code for Freescale Coldfire microprocessors and the Motorola 680000 (68K) family is supported.
- **C/C++:** *Understand* analyzes K&R or ANSI C source code and most constructs of the C++ language. *Understand* works with any C compiler, and has been tested with most of the popular ones. Note that C++ templates are not yet supported.
- **Objective C/Objective C++:** *Understand* provides a strict analyzer option that supports these languages.
- **C#:** *Understand* supports C#.
- **COBOL:** *Understand* supports the Ansi85, Micro Focus, AcuCobol, and IBM compilers.
- **FORTRAN:** *Understand* supports FORTRAN 77, FORTRAN 90, FORTRAN 95, and FORTRAN 2003, in both free and fixed format. Extensions supported include Harris FORTRAN and DEC FORTRAN. We often expand *Understand* to support common compiler extensions. If you find that the compiler extensions you are using are not currently supported, contact us at support@scitools.com.
- **Java:** *Understand* supports most of JDK 1.3, 1.4, 5, and 6. Specifically, the generics introduced in JDK 5 are not currently supported. Source code containing generics may be analyzed but generics information will be ignored.
- **JOVIAL:** JOVIAL73 and JOVIAL3 are supported.
- **Pascal:** *Understand* supports all versions of Borland's Delphi language and Borland's Turbo Pascal language. It also supports ISO 7185: 1990 (also known as Unextended Pascal) with DEC Pascal extensions. You can also enable support for Ingres embedded SQL statements.
- **PL/M:** The standard version for PL/M 80/86 is supported.
- **Python:** *Understand* supports both Python 2.x and 3.x.
- **VHDL:** Versions VHDL-87, VHDL-93, and VHDL-2001 are supported.
- **Web:** HTML, PHP, CSS, Javascript, and XML files are supported.

For information about support for a specific language syntax, search the Build Log on the Scientific Toolworks website (<http://www.scitools.com/support/buildLogs.php>) and the Forum (<http://www.scitools.com/support/forum/>).

For Those Who Don't Like to Read Manuals

If you are like many engineers at Scientific Toolworks, you like to just dig in and get going with software. We encourage that, or at least we are pragmatic enough to know you will do it anyway! So feel free to use this manual as a safety net, or to find the less obvious features. However, before you depart the manual, skim the next chapter for tips on effectively utilizing what *Understand* has to offer.

Here are some places other than this manual to look for advice:

- Use the links in the Getting Started display (**Help > Getting Started** from the menus)
- Choose **Help > Help Content** from the menus.
- Use **Help > Example Projects** to play with sample code.
- Choose **Help > Frequently Asked Questions** to see the FAQ list on our website.
- Choose **Help > View SciTools Blog** to read the blog on our website. You can get the latest blog topics by clicking the **Refresh** button.
- For videos of various features, see <http://www.scitools.com/support/videos.php>.

For more advanced users, try these information sources:

- Choose **Help > About Understand** to see which build you are currently running.
- See <http://www.scitools.com/support/buildLogs.php> to search through the build logs. Use the link on that page to “Sign up to receive via Email” new build notes and build announcements.
- See <http://www.scitools.com/support/forum/> to read and ask questions in our Forum.
- Choose **Help > Key Bindings** for keystroke help.
- See <http://www.scitools.com/documents/metrics.php> for details about specific metrics.
- Choose **Help > Perl API Documentation** and **Help > Python API Documentation** for help on scripting.

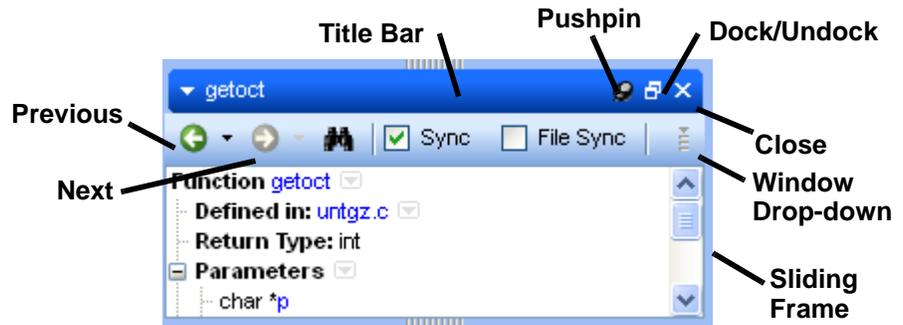
This chapter helps you put *Understand* to good use quickly and easily by describing the basic windows in *Understand*.

This chapter contains the following sections:

Section	Page
Using Understand Windows	18
Understand Terminology	19
Starting Understand	21
Context Menus Are Everywhere	23
Quickly Find Things in Your Source	25
Information Browser	28
Source Editor	29
Architecture Browser	30
Graphical Views	31
ASCII and HTML Reports	32
APIs for Custom Reporting	32

Using Understand Windows

Understand has a main window and many smaller areas that open within the *Understand* application window. You can arrange these areas in your workspace to meet your needs.



- **Title Bar:** You can drag the title bar of an area around the main window. If you move to the edge of the main window, a docking area expands. If you drop the area there, it “docks” to the edge of the main window.
- **Pushpins and Drawers:** Click the  icon to move an area to a tab along the same edge of the main window to which this area was docked. This is a “drawer” that opens automatically if you point your mouse at the tab title. The drawer closes if you move your mouse away from the area without clicking on it or if you click the title tab of the currently open drawer.
Click the  icon to “pin” a drawer open. Pinned drawers have a title bar and title bar icons like the ones shown above.
- **Dock/Undock:** Click the  icon to change the area to an undocked window. Click the icon again in an undocked window to return to a docked area.
- **Close:** Click the “X” icon to close the area or undocked window.
- **Drop-down:** Click the  icon to see the context menu for this area. Right-clicking an item within an area usually displays a context menu specific to that item.
- **Sliding Frame:** You can drag the frames between window areas to change their sizes.
- **Previous and Next:** Each area type has different icons below the title bar. For the Information Browser area shown, you can browse through the history of entities viewed. For other areas, you will see other icons.

Understand Terminology

Before continuing with the rest of this manual, please take a moment to familiarize yourself with *Understand's* terminology. Doing so will make reading the manual more helpful and put you on the same sheet of music as the technical support team should you need to email or call.

Architecture: An architecture is a hierarchical aggregation of source code units (entities). An architecture can be user created or automatically generated. Architectures need not be complete (i.e an architecture's flattened expansion need not reference every source entity in the database), nor unique (that is, an architecture's flattened expansion need not maintain the set property).

Database: The database is where the results of the source code parsing, as well as project settings, are stored. By default, this is a project's ".udb" file.

Entity: An *Understand* "entity" is anything it has information about. In practice this means anything declared or used in your source code and the files that contain the project. Subroutines, variables, and source files are all examples of entities.

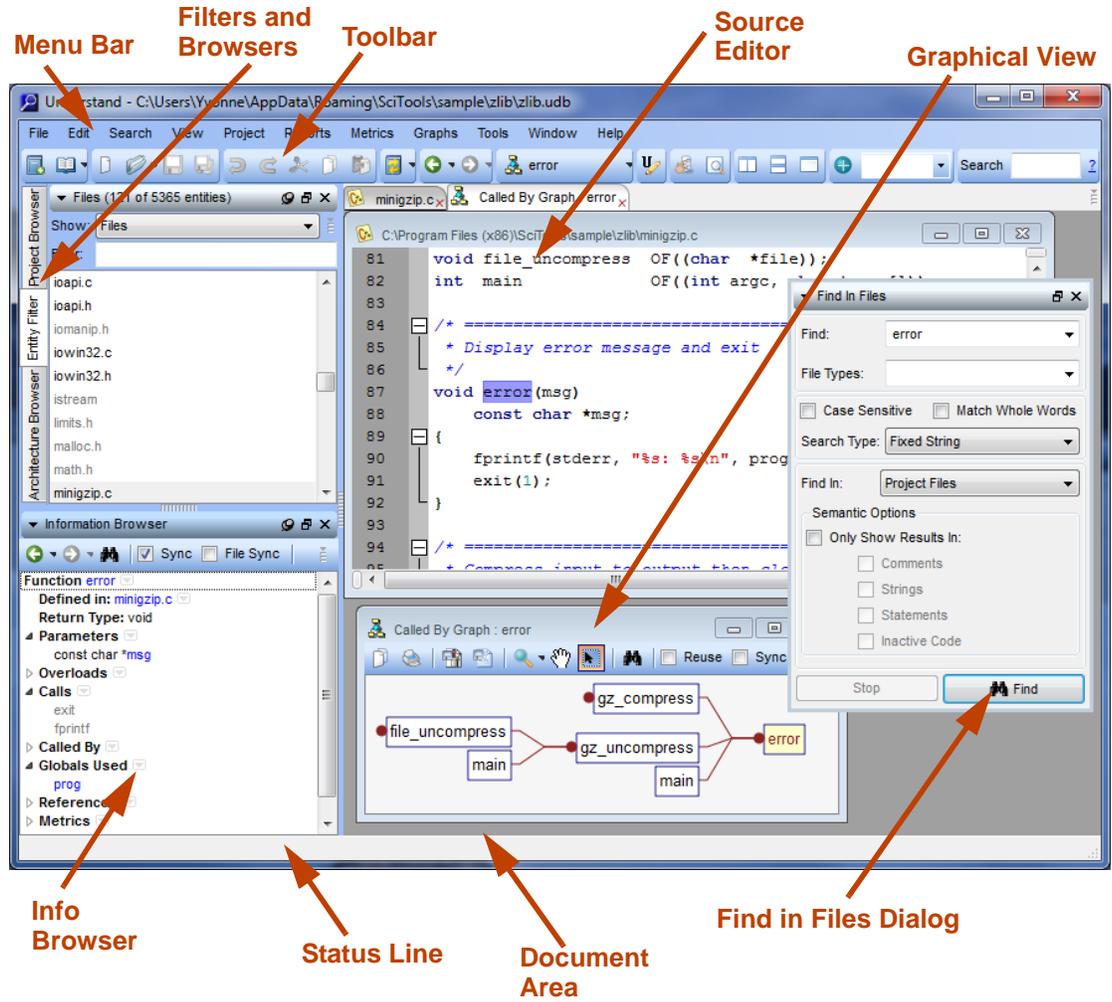
Project: The set of source code you have analyzed and the settings and parameters chosen. A "project file" contains the list of source files and the project settings.

Relationship: A particular way that entities relate to one another. The names of relationships come from the syntax and semantics of a programming language. For instance, subroutine entities can have "Call" relationships and "CalledBy" relationships.

Script: Generally a Perl script. These can be run from within *Understand's* GUI, or externally via the "uperl" command. The *Understand* Perl API provides easy and direct access to all information stored in an Understand database.

Parts

The following figure shows some commonly used main parts of the *Understand* graphical user interface (GUI):

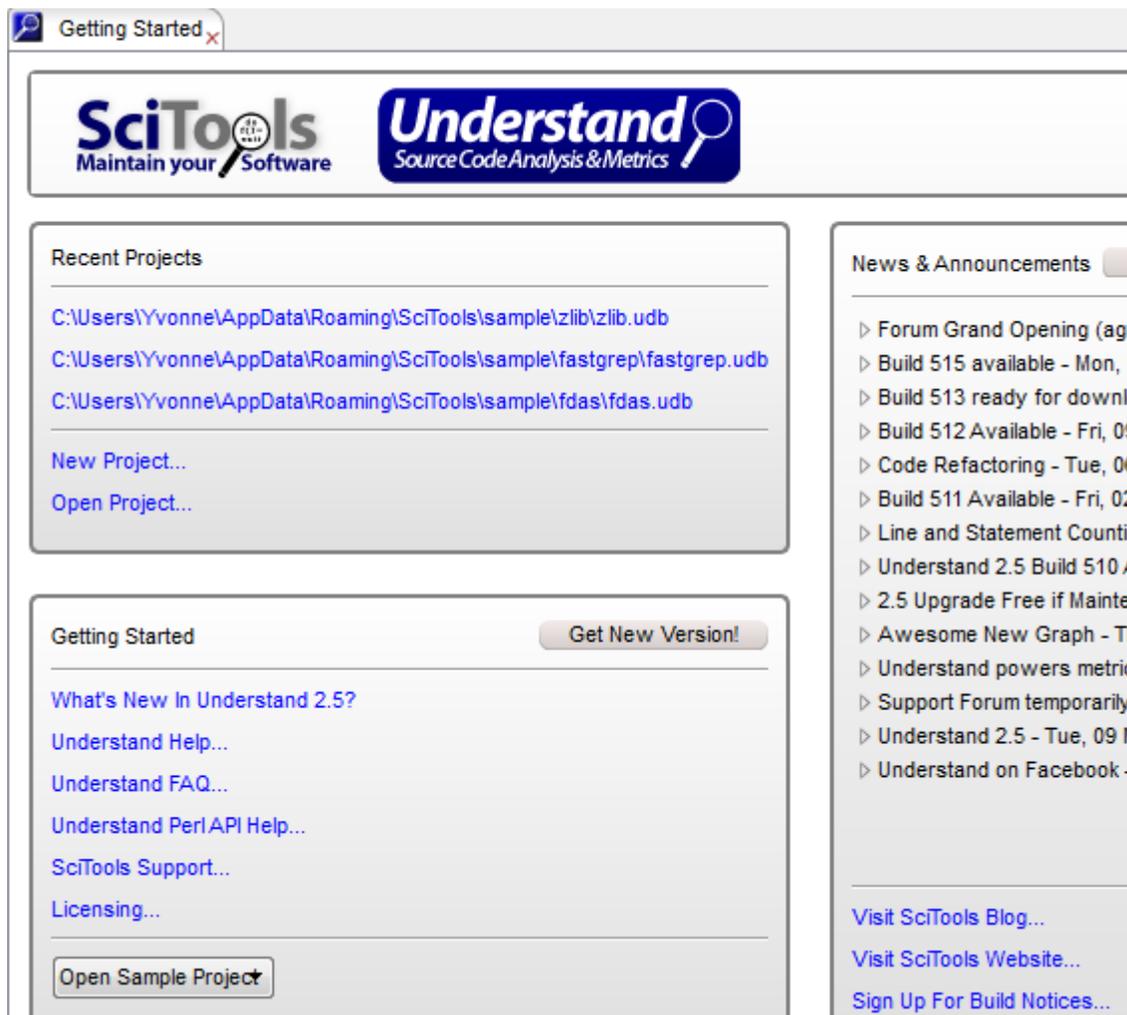


Starting Understand

When you install *Understand* on Windows, a command to launch the software is added to your Windows **Start** menu in the SciTools folder.

When you start *Understand*, you see the **Getting Started** tab in the Understand window. To begin creating a new project, click **New Project...** and see *Creating a New Project* on page 35 for details.

If you've used a project recently, it is listed in the Getting Started tab, and you can click to open it. If the existing project you want to open isn't listed, click **Open Project...** and browse for it.



You can also choose **File > Open > Project** and **File > Recent Projects** from the menus to open projects.

If you are learning about *Understand*, use the links in the **Getting Started** box. You can click **Open Sample Project** and choose an example project that uses a source code language used in your own projects.

If you are a more experienced *Understand* user, use the links in the **News & Announcements** box to keep your knowledge current.

If you have closed the Getting Started tab and want to reopen it, choose **Help > Getting Started** from the menus. If you don't want to see the Getting Started tab every time you run *Understand*, uncheck the **Show on Startup** box.

When you are finished using a project, you can open another project or choose **File > Close <project_name>.udb**. You will be asked if you are sure you want to close the project. If you have made any changes to files, you will be prompted to save or discard the changes for each file individually.

If you want to make sure you have installed the latest version of *Understand*, you can choose **Help > Check for Updates** from the menus. (You'll see the **Get New Version** button in the Getting Started tab if a new version is available.)

Other Ways to Run Understand

For information on running *Understand* from the command line, see Chapter 14, Command Line Processing.

If multiple users will run *Understand* from the same Windows machine, each user may have a separate initialization file. These files store user preferences. *Understand* looks for the initialization file location in the following locations, depending on the operating system (on Windows, this location is referenced by the WINDIR environment variable):

- **Windows 2000/XP:** C:\Documents and Settings\USERID\Application Data\SciTools\Understand.ini
- **Windows Vista/7:** C:\Users\USERID\AppData\Roaming\SciTools\Understand.ini
- **Linux/Unix:** ~/.config/SciTools/Understand.conf
- **Mac OS X:** ~/Library/Preferences/com.scitools.Understand.plist

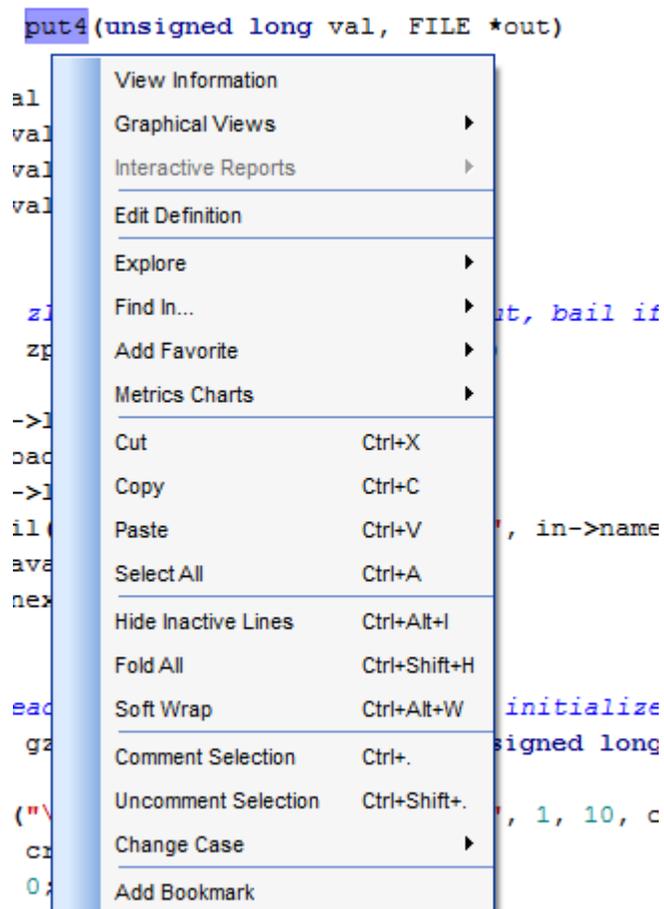
Context Menus Are Everywhere

Right-clicking gets you a long way in *Understand*; almost everywhere you point, you can learn more and do more by bringing up menus with your right mouse button.

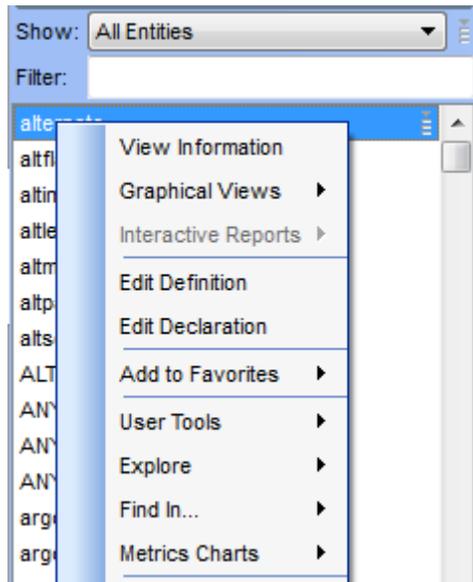
Tip: Hold down the Ctrl key while right-clicking to create new windows rather than re-using existing ones.

Remember to right-click, anytime, anywhere, on any entity to get more information about that entity.

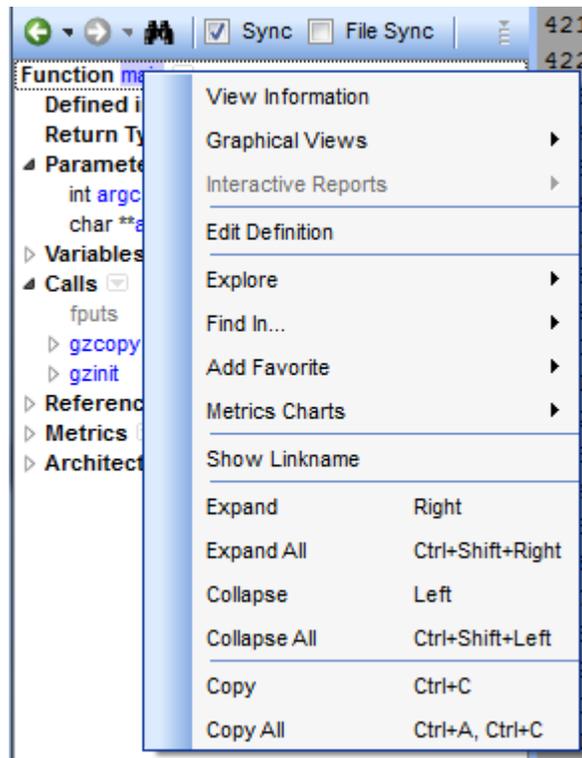
Example: Right-click on an entity in the Source Editor:



Example: Right-click on an entity in the filter area:



Example: Right-click on an entity in the Information Browser:



Quickly Find Things in Your Source

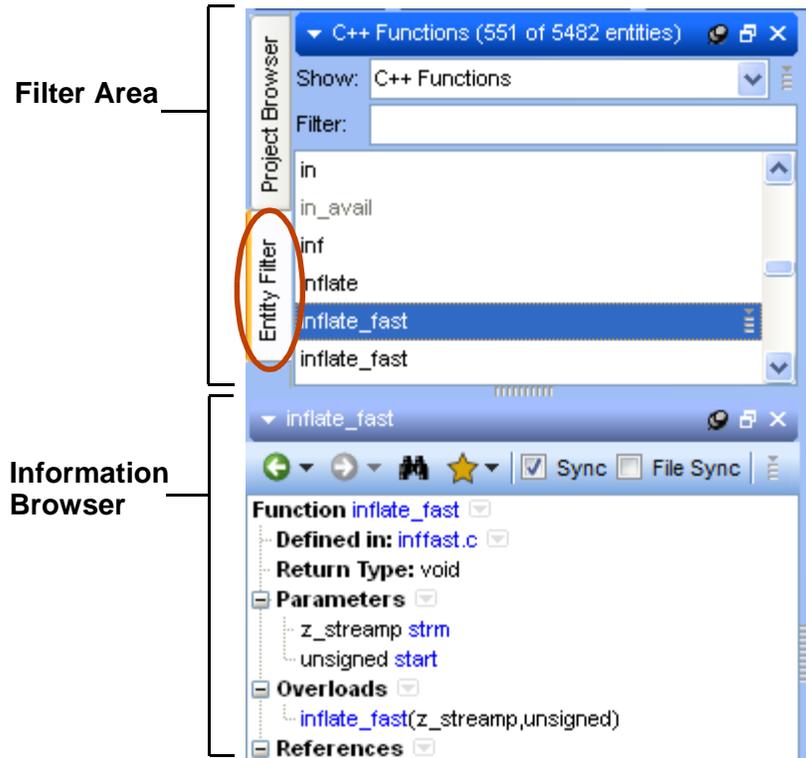
Understand provides several ways to quickly locate items of interest in your source code. These features include the Filter Area, the Entity Locator, and the Find in Files dialog.

Entity Filter

The filter area of the *Understand* window helps you quickly find things in your code by separating that database into lists of Files, Classes, Functions, Objects, Types, Macros, Subprograms, Packages, Modules, Blocks, Methods, Interfaces, SQL Tables, and more. The types of filters available depend on the languages you have configured your *Understand* project to understand.

After clicking in the filter area, you can type a letter to move to the first entity beginning with that letter in the current list.

By default, the *Information Browser* shows all known information about the selected entity. It is a key to navigating in *Understand*.

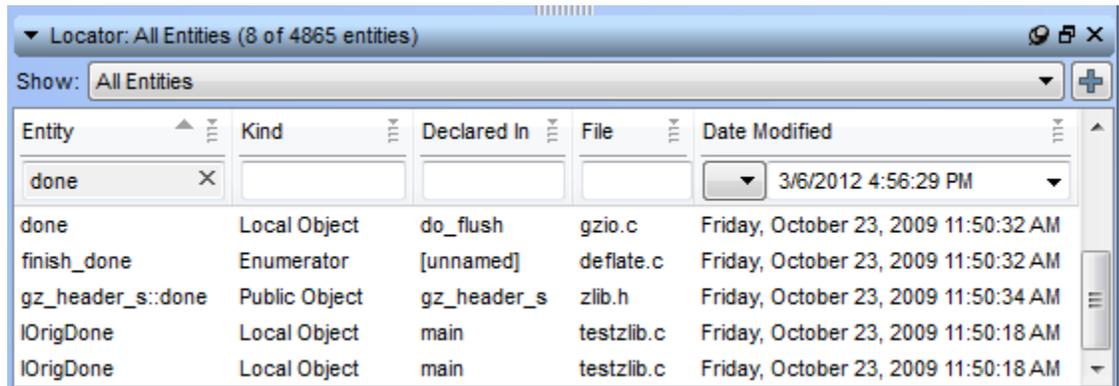


For details, see *Entity Filter* on page 123 and *Information Browser* on page 125.

Entity Locator

The filter provides a quick way to find major items that were declared and used in your project. However, some items such as local parameters, variables, and unresolved variables (used but not declared in the processed source) are not listed in the filters. To search or browse the entire database for your project, use the *Entity Locator*.

To open the *Entity Locator*, choose **View > Entity Locator**.



By default, this area lists all the entities in the project. You can search for entities matching a particular text or regex string using the fields above each column.

For details, see *Entity Locator* on page 148.

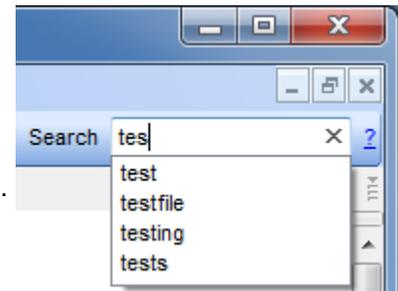
As in any other window, the context menu is also active.

You can select multiple rows and columns and copy their contents to the clipboard. When you paste, the contents will be pasted as tab-separated text.

Instant Search

Instant Search lets you search your entire project instantly, even if it contains millions of lines of source code. As you type, you can see terms that match the string you have typed so far.

A number of powerful search options are supported with Instant Search. See *Instant Search* on page 141.



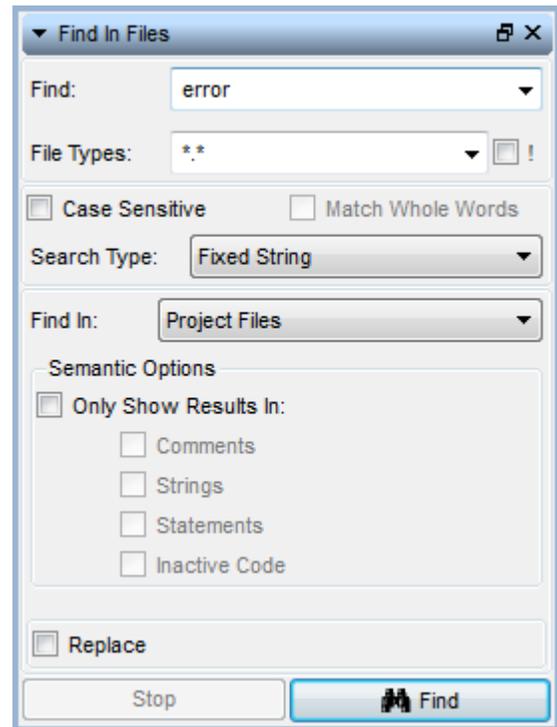
Find in Files

Similar to the UNIX command *grep*, you may search files for the occurrence of a string. Select **Find in Files** either from the **Search** menu or from a context menu.

When you click **Find**, a list of all occurrences matching the specified string or regular expression is displayed in the *Find Results* window. Double click on any result to display the *Source View* where the string occurs.

The options let you set behaviors such as case-sensitivity and wildcard pattern matching.

See *Find in Files* on page 143 for more information.



Favorites

You can place entities and code locations that you often use on your Favorites list. To add a favorite, right-click on it and select **Add to Favorites** and the name of the list to contain this item.

To see the Favorites list, choose **View > Favorites** and the name of the list to open.



See *Favorites* on page 135 for more information.

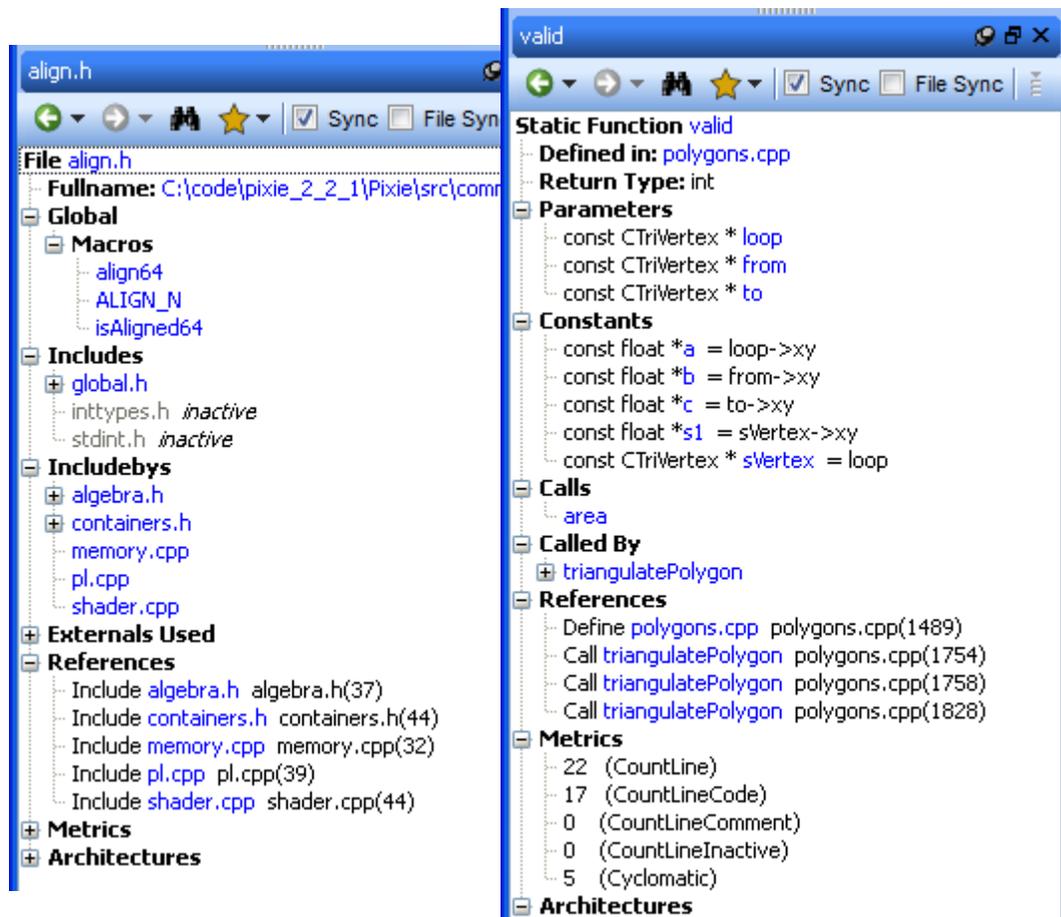
Information Browser

Just about everything *Understand* knows about code is shown in the Information Browser (IB). The IB is used for all types of entities.

The Information Browser shows different things depending on the type of entity selected.

It shows different kinds of information about entities such as source files, classes, members, functions, types, methods, packages, interfaces, and more. Information that is hierarchical in nature (such as a call relationship) can be expanded multiple levels.

Below are Information Browser windows for a file and a C++ function:



For details, see *Information Browser* on page 125.

Source Editor

Understand has a source editor that not only lets you edit your source code, it colorizes the source code and tells you about the code you are editing.

Source can be visited by double-clicking almost anywhere else in the tool. You can move forward or backward through such “visits” by using the **Next** and **Previous** icons in the toolbar.

```

zlib\infrees.h*
43  #define ENOUGH 2048
44  #define MAXD 592
45
46  /* Type of code to build for inflate() */
47  typedef enum {
48      CODES,
49      LENS,
50      DIST
51  } codetype;
52
53  extern int inflate_table OF((codetype type,
54      unsigned short FAR *lens,
55      unsigned codes, code FAR * FAR *table,
56      unsigned FAR *bits, unsigned short FAR *work));
57
Line: 53 Column: 25 | RW | C++ | TrackBack 1 Rev

```

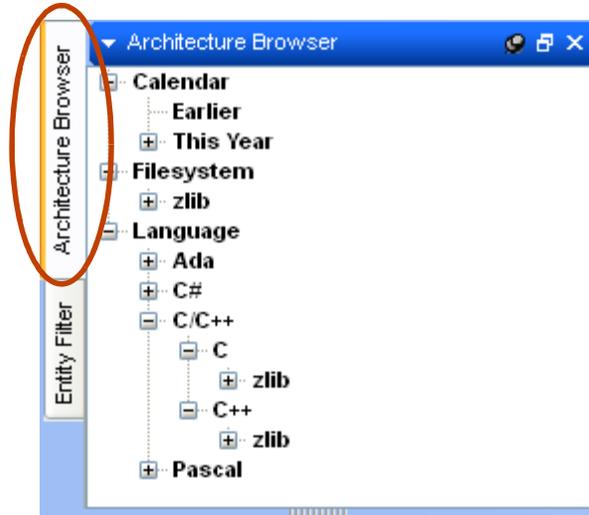
As with any other place in *Understand*, a context menu is available throughout the editor. To learn about something just right-click on it to see what information is available.

For details, see *Source Editor* on page 158.

Architecture Browser

The Architecture Browser allows you to manage *architectures*. It shows a list of all the defined architectures in the database and provides a way to navigate individual architectures.

For example, this window shows the auto-architectures provided with *Understand*: Calendar, Directory Structure, Languages. The architectures are expanded somewhat here to show the top-level nodes for an example application.



You can use the auto-architectures, create your own architectures, import and export architectures (as XML files), generate graphs and metrics for any level in an architecture hierarchy, and combine architectures through filtering.

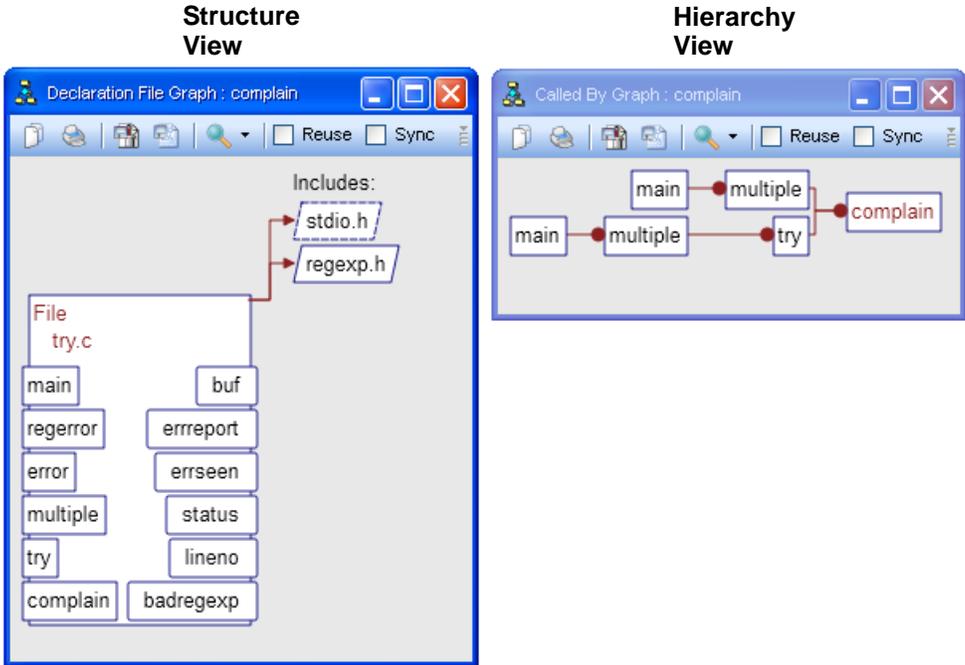
For details, see *About Architectures* on page 176.

Graphical Views

Understand analyzes your software code and creates a database containing information about the entities and the relations between entities. The database can then be browsed using various “graphical view” windows. The graphical views are divided into these kinds:

- **Hierarchy** views show relations between entities. Each view follows a relation (for instance “Calls”) from the starting entity (that you inquired about) through its children and successors.
- **Structure** views quickly show the structure of any entity that adds to the structure of your software (for instance a package, function, procedure, or task).

Examples of each type are shown in the following figure:



For details, See *Using Graphical Views* on page 228.

ASCII and HTML Reports

Views in *Understand* provide information about individual entities. The reports bundle information about all entities in ASCII or HTML format.

Understand

Table of Contents

[\(Index\)](#)

[Data Dictionary](#)

[File Contents](#)

[Program Unit Cross Reference](#)

[Object Cross Reference](#)

Program Unit Cross Reference

Non-Alpha	A	B	C	D	E	F	G	H	I	J	K	L	M	N
---------------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	-------------------

[Deleg_foo](#) (Function)
Declared as: Void
Define [\[Deleg.c, 9\]](#) Deleg.c

[Deleg_Instance_init](#) (Function)
Declared as: Void
Define [\[Deleg.c, 4\]](#) Deleg.c

The HTML and ASCII reports also show information not available interactively, such as project metrics and quality reports. These reports are suitable for printing or browsing with a web browser.

See *Using Reports* on page 190 for more information.

APIs for Custom Reporting

Understand data is also available directly from scripts and programs that you (or we) write. A C API (usable from C, C++ or other languages that can call C libraries), a Python interface, and a Perl interface are provided with *Understand*.

Using the API, you have exactly the same access that we have when we write the existing GUI and report generators.

This manual doesn't cover the APIs. Choose **Help > PERL API Documentation** or **Help > Python API Documentation** for more information.

The **Reports > Project Interactive Reports** and **Graphs > Project Graphs** commands display a list of user-created plugins, which can be created using the Perl API. For information about creating plugins, please contact support@scitools.com. The SciTools forum at <http://scitools.com/support/forum> and the SciTools blog at <http://scitools.com/blog> also contain messages concerning plugins.

This chapter shows how to create new *Understand* project files that you will use to analyze your source code.

This chapter contains the following sections:

Section	Page
About Understand Projects	34
Creating a New Project	35
Project Configuration Dialog	39
Languages Category	41
Files Category	42
File Types	48
File Options	49
Scheduled Activities	50
Metrics	52
Reports	54
Visual Studio	57
Annotations	59
Ada Options	61
Assembly Options	65
COBOL Options	67
C++ Options	69
C++ (Strict) Options	76
C# Options	79
FORTRAN Options	80
Java Options	83
JOVIAL Options	85
Pascal Options	87
PL/M Options	89
Python Options	91
VHDL Options	91
Web Options	92
Setting General Preferences	93
Analyzing the Code	114
Converting an Understand 1.4 Project	119

About Understand Projects

Understand is like a compiler, except it creates information, not executable code.

In order for *Understand* to analyze your source code, it needs much of the information your compiler needs. It needs to know:

- What source files to analyze
- The type of source code
- The standard library paths and include directories
- Where to find Java .jar files that provide classes for which you do not have source code
- Compiler/environment specific macros that need to be defined for the pre-processor
- Application-specific macro definitions
- What implementation parameters (such as integer precision) and column truncation settings to use
- Any namespaces

If you developed the program or have been working with it for some time, this information is probably obvious to you. However, if you inherited this source code from another programmer, team, or company, you will probably have to examine the project building files (for example, a makefile) in order to come up with the information needed for accurate parsing of the code.

The easiest way to analyze your code is to use *Understand's* GUI to build and parse a project. This chapter will walk you through that process.

The *Understand* Project Database

The *Understand* project database is stored in a proprietary binary format. The file format uses a network/object format that is optimized for storing *Understand* information.

Understand databases have a file extension of **.udb**.

The project file permits multiple simultaneous read accesses, but it does not support multi-user write access.

Occasionally, a new feature to *Understand* requires a change to the database format. Such changes are noted in the Change Log. When you install a build that modifies the database format, existing projects are automatically reparsed when you open them.

Creating a New Project

To begin analyzing code, you create a project and specify what source files to parse. *Understand* parses your code and creates a database you can browse. This database can be refreshed incrementally in the GUI or updated using command-line tools.

This section shows how to create a new project. The project will be stored in a *Project Database*, which has a file extension of `.udb`.

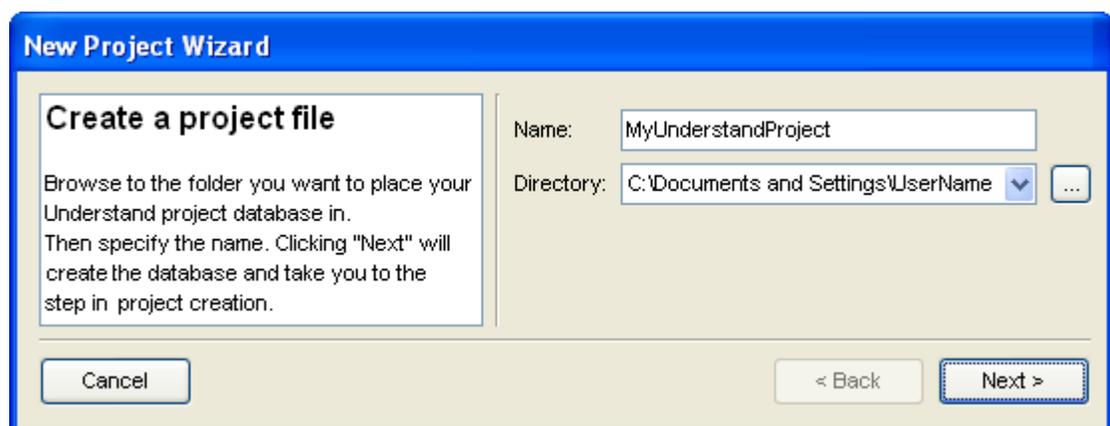
To create a new project, follow these steps:

- 1 Click the **New Project** link in the Getting Started tab that you see when you start *Understand*. Or, choose **File > New > Project** from the menus.
 - By default, this opens the New Project Wizard, which is described on page 35.
 - Alternately, you may have disabled the option to run this wizard, in which case, you see the "Create new project as..." dialog. Browse to the folder where you wish to create the project database. Type the name of the project in the **File name** field. A `.udb` file extension will be added automatically. Click **Save**. You will see the Understand Project Configuration dialog, which is described in page 39.
 - Another way to create a project is to add Buildspy to your gcc/g++ build process. This automatically generates an *Understand* project when you compile your project. See *Using Buildspy to Build Understand Projects* on page 306.

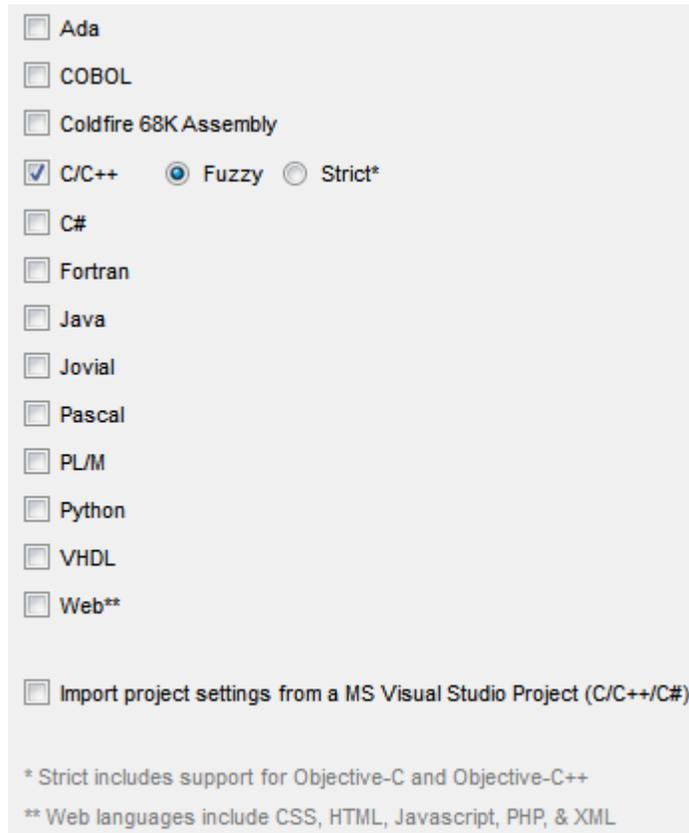
New Project Wizard

Unless you have disabled the New Project Wizard, this is the tool you use to create projects. To open it, click the **New Project** link in the Getting Started tab that you see when you start *Understand*. Or, choose **File > New > Project** from the menus.

- 1 In the Create a Project File page of the wizard, type a **Name** for the project and browse for a directory to contain the *Understand* project files. It is often handy to have the project file in the top-level directory of the source code, but this is not required. If the directory does not exist, you are asked if you want it created.



- 2 Click **Next** to see the Languages page of the wizard.



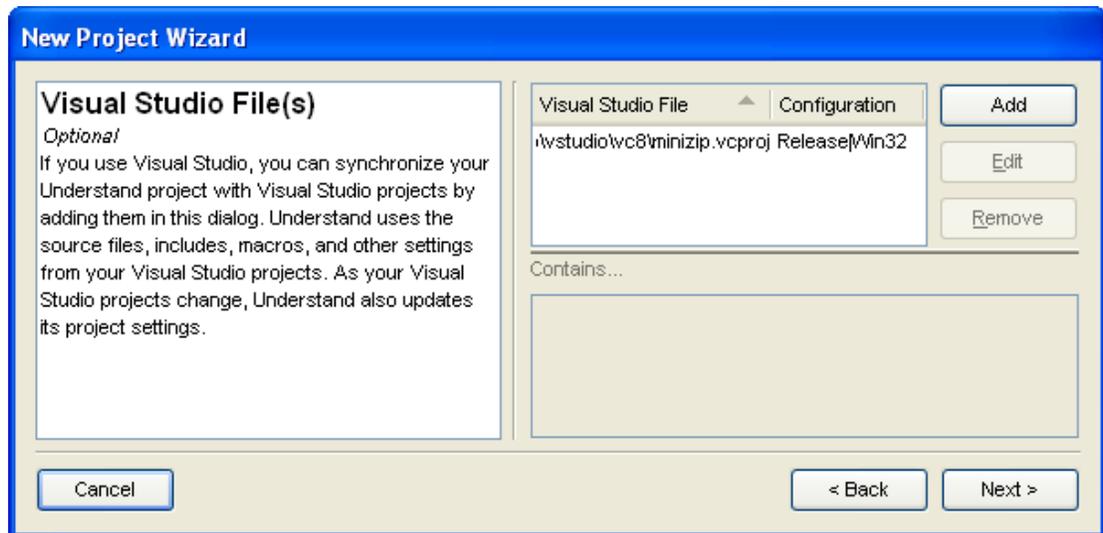
Ada
 COBOL
 Coldfire 68K Assembly
 C/C++ Fuzzy Strict*
 C#
 Fortran
 Java
 Jovial
 Pascal
 PL/M
 Python
 VHDL
 Web**

Import project settings from a MS Visual Studio Project (C/C++/C#)

* Strict includes support for Objective-C and Objective-C++
** Web languages include CSS, HTML, Javascript, PHP, & XML

- 3 Put checkmarks next to languages used in the source code for this project. See *Languages Category* on page 41 for more information about specific languages and the strict C/C++ option.
- 4 If you use Microsoft Visual C for your C, C++, or C# code, you can check the box to import project settings. Then click **Next**.

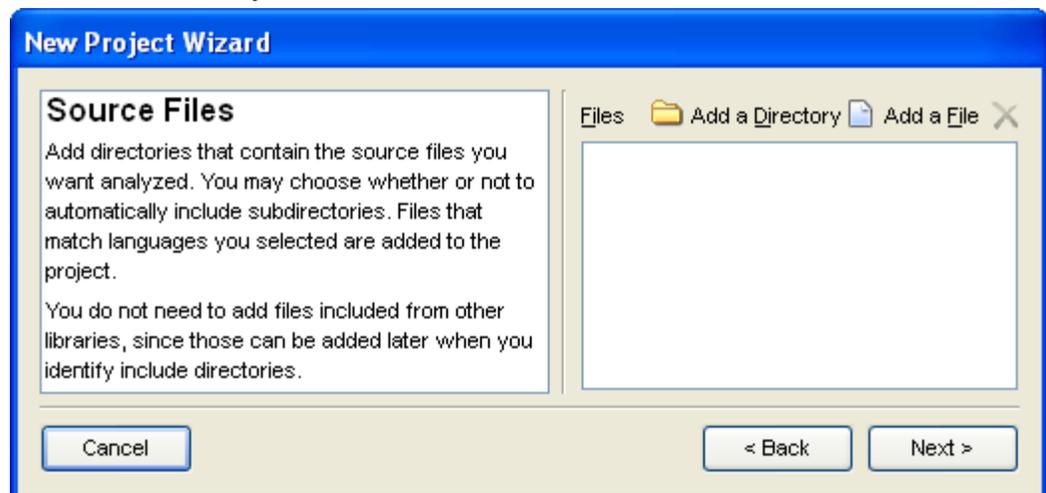
- 5 If you checked the **Import project settings from a MSVC Project** box, you see the Visual Studio File(s) page. Otherwise, skip to the next step.



To synchronize your Understand project with Visual Studio projects, click **Add**. In the Add a new Visual Studio file dialog, click ... and browse for your Visual Studio project file. In the Add a new Visual Studio file dialog, select the project configuration you want used when *Understand* analyzes your project. Then click **OK**. See *Visual Studio* on page 57 for more information.

You can add multiple Visual Studio projects or use the **Edit** button to change the Configuration setting. Then click **Next**.

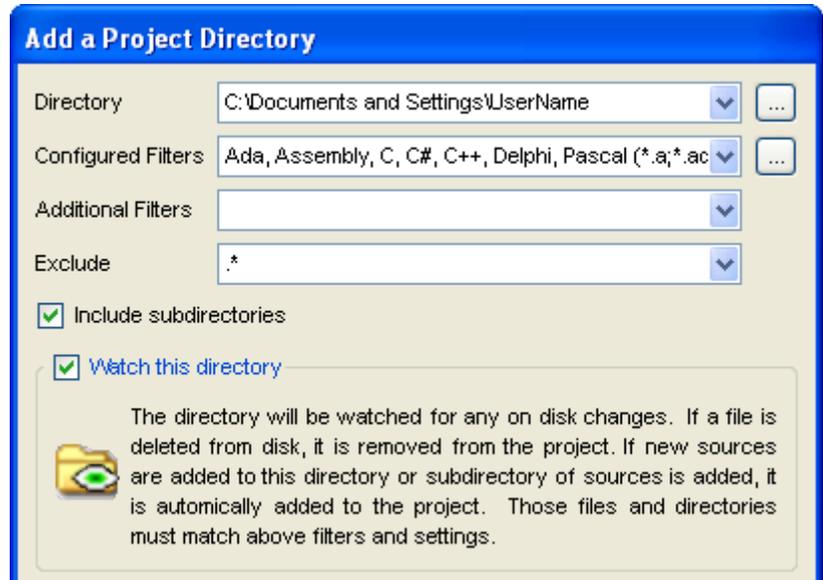
- 6 In the Source Files page of the wizard, you add source files to a project by clicking **Add a Directory** or **Add a File**.



To add a file, just browse for the file and add it.

When you add a directory, you can browse for a directory, modify the list of languages used in the source files, add additional filters for file extensions not expected by *Understand*, filter out any files you want to exclude (for example,

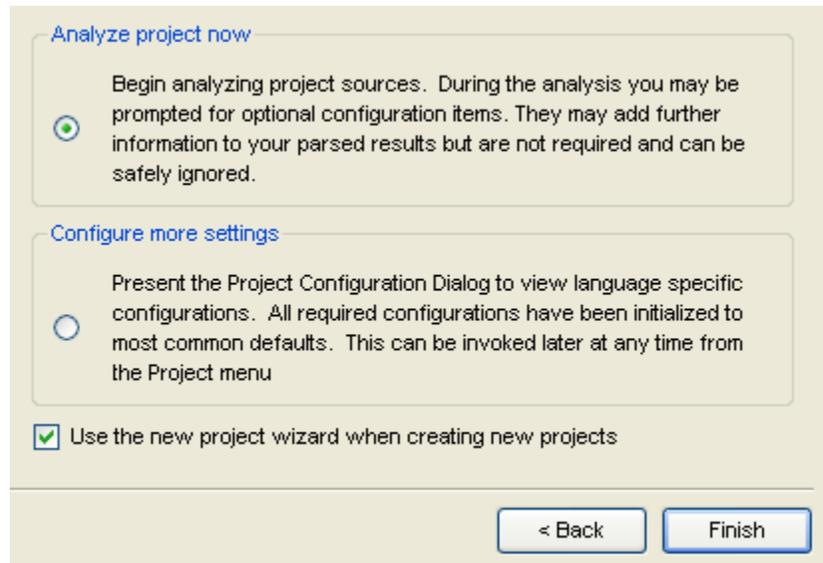
temp*.*)), and choose whether all the subdirectories of this directory should be added. You can also choose whether the directory will be watched for changes. See *Adding Directories* on page 43 for details.



If you chose a Visual Studio project, those files are automatically listed in the Source Files page of the New Project Wizard.

If you want to delete a file or a directory (and its subdirectories), select that item and click the “X” icon.

- 7 Choose whether to **Analyze project now** or further configure the project. Choosing **Configure more settings** takes you to the Project Configuration dialog, which is described starting on page 39. In either case, you can go to the Project Configuration dialog anytime you like.

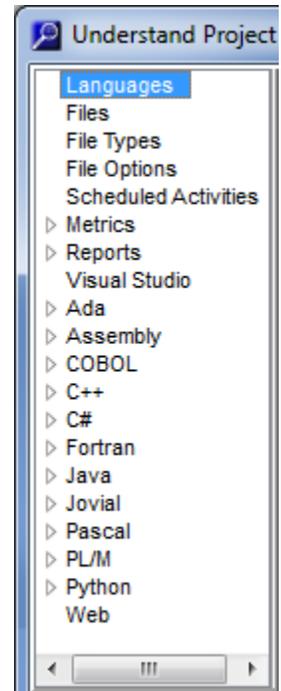


Project Configuration Dialog

The Understand Project Configuration dialog opens when you create a new project or when you choose the **Project > Configure Project** menu item.

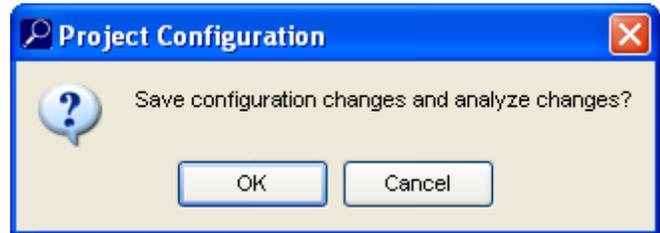
The categories on the left in the Project Configuration dialog allow you to specify various project settings to be used during analysis. The Project Configuration dialog contains the following categories:

- **Languages:** Specify the types of languages to be parsed. For details, see *page 41*.
- **Files:** Specify the locations of source files to be analyzed. For details, see *page 42*.
- **File Types:** Specify how to handle source file types and what file extensions are used. For details, see *page 48*.
- **File Options:** Specify the file encoding and editing mode for source files. For details, see *page 49*.
- **Scheduled Activities:** Schedule events to take place at regular intervals. For details, see *page 50*.
- **Metrics:** Specify the metrics you want computed for this project. For details, see *page 52*.
- **Reports:** Specify the reports you want generated for this project. For details, see *page 54*.
- **Visual Studio:** Specify a Visual Studio project to synchronize this *Understand* project with. For details, see *page 57*.
- **Language-Specific Options:** Specify options for the languages you selected in the Languages category. For details, see:
 - Ada Options, *page 61*
 - Assembly Options, *page 65*
 - COBOL Options, *page 67*
 - C++ Options, *page 69*
 - C++ Strict Options, *page 76*
 - C# Options, *page 79*
 - Fortran Options, *page 80*
 - Java Options, *page 83*
 - JOVIAL Options, *page 85*
 - Pascal Options, *page 87*
 - PL/M Options, *page 89*
 - Python Options, *page 91*
 - Web Options, *page 92*



For advice about ways to adjust the project configuration to improve the accuracy of project analysis, see the [SciTools Blog](#).

After you change the project configuration, click the **OK** button and the configuration will be saved. Whenever you modify the files in the project configuration, including at the time of project creation, a dialog alerting you to the change in configuration appears.

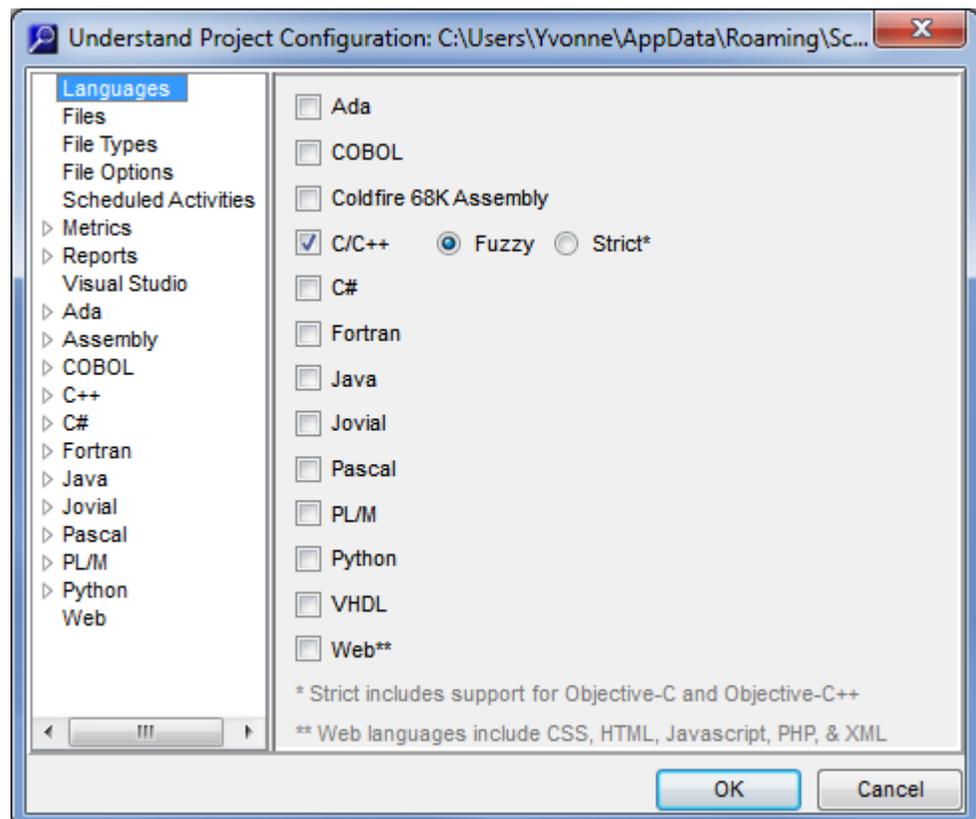


Click **OK** and *Understand* begins parsing (that is, analyzing) the code.

If you want to close the Project Configuration dialog without saving any changes, click **Cancel**, and then click **Yes** in the box that asks if you really want to cancel changes.

Languages Category

In the **Languages** category of the Project Configuration dialog, you can check boxes for the languages used in your project. A project can contain source code in one or more languages.



When you select a language, categories for that language are added to the list on the left in the Project Configuration dialog. The languages you choose here not only affect how the source files are parsed. They also affect the filter types available, the metrics available, and the report types available.

If you select multiple languages, references between those languages are analyzed. For example, if C code calls a Java function, that reference will be found.

If you have C or C++ code, you can decide to use either the default C/C++ parser (the “fuzzy” analyzer) or the newer “strict analyzer”. To use the strict analyzer, check the **Strict** option next to C/C++. Internally, checking this box causes a completely separate parser to be used to analyze your C/C++ code.

The “strict analyzer” provides the following features:

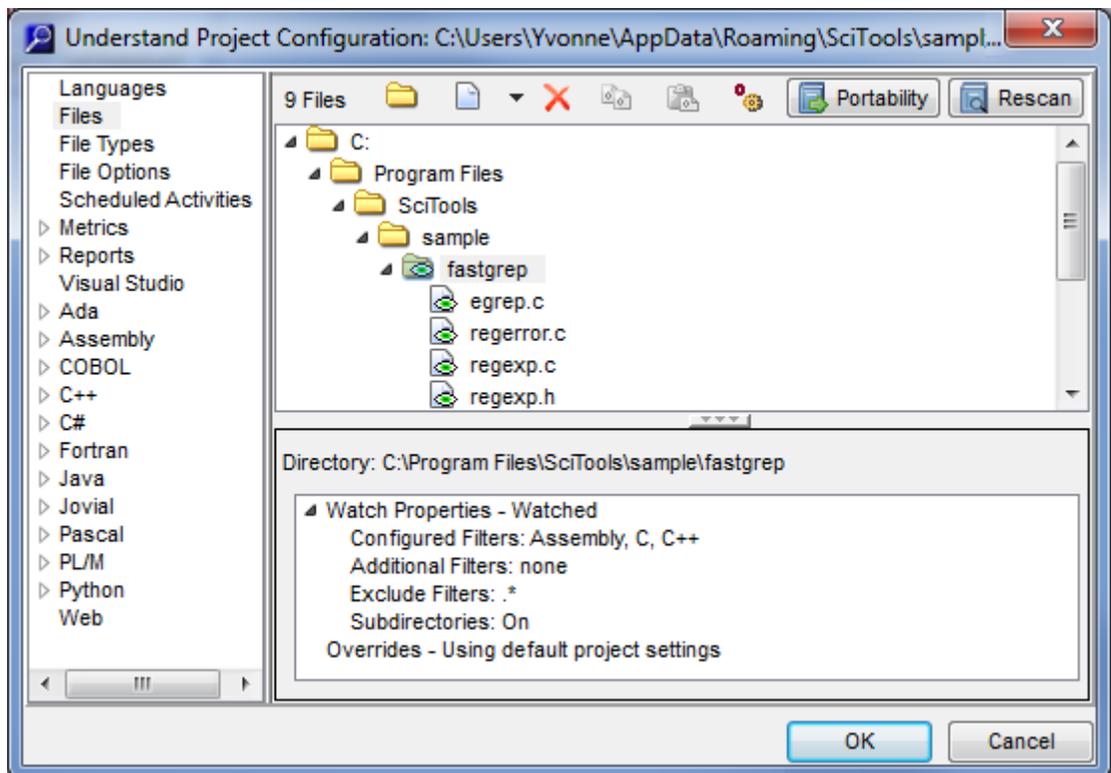
- Support for Objective-C and Objective-C++ (used for Mac OS and iOS) is provided with the strict analyzer, but not with the default C/C++ parser.
- Provides better support for Templates.
- Provides better support for Overloaded functions.

The default C/C++ parser aims to use fuzzy logic to handle incomplete, non-compiling code gracefully and as accurately as possible. The new parser is more strict than the old parser and requires a more accurate project definition—for example, by specifying all include paths and macro definitions and including only those files in the project that are used in the software build. For details, see the [Improving Project Accuracy \(C/C++\)](#) blog post.

If you are using the strict analyzer, see *C++ (Strict) Options* on page 76 for how to configure your project.

Files Category

In the **Files** category of the Project Configuration dialog, you can add source code directories and/or individual files to the project. You can also delete specific files from the analysis and modify the language-specific options for individual directories and files.



You can add source files here, or you can tie the project to those specified in an MS Visual Studio project file (MS Windows versions of *Understand* only). See *Visual Studio* on page 57.

The top area shows the directories and files you have added in a tree that you can expand. It also shows how many files are currently in the project.

The bottom area shows any option overrides you have set for the selected directory or file.

Icons at the top of the dialog perform the following actions:

-  Open the Add a Directory dialog.
-  Open the Add a File dialog.
-  Choose Add a File or import a list of files
-  Delete the selected directory or file from the project analysis.
-  Copy the override settings for the selected directory or file.
-  Paste the override settings to the selected directory or file.
-  Configure override settings for the selected directory or file.

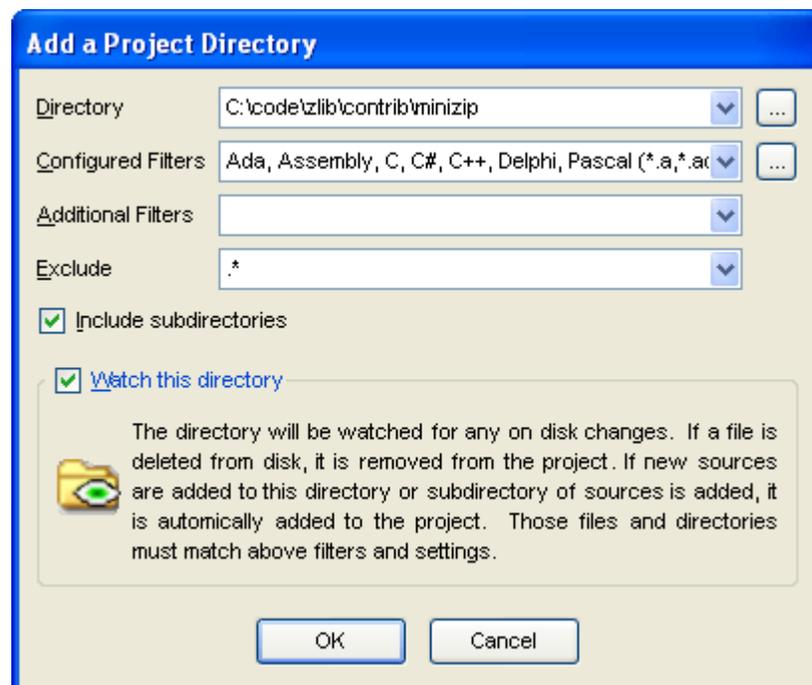
Note that your changes are not saved until you click **OK**.

Click **Portability** to set portability options for file paths. See page 47.

Click **Rescan** if you have added files to a directory that are not shown in the tree in this dialog.

Adding Directories

To add source directories to the project, click . You see the Add a Directory dialog:



- 1 In the **Directory** field, type the full directory path. Or, you can click the ... button and use the Browse for Folder dialog to locate a directory containing source files and click **OK**.

- 2 In the **Configured Filters** field, click the ... button if you want to add or delete languages from the list shown. In the Select Filters from Configured File Types dialog, put a checkmark next to any languages you want to be recognized as part of the project. Notice that additional languages are listed beyond those shown in the Languages category. These include JavaScript, MSdos Batch, Perl, Tcl, Text, and XML.

If this directory contains source files with extensions that are not listed, click **Configure**. Also, see *File Types* on page 48. For example, you might add .a64 as an assembly file type.

- 3 In the **Additional Filters** field, type a pattern-matching string that matches only the files you want to keep in the analysis. For example, std*.* includes only files that begin with “std”. You can separate filters with a comma.
- 4 In the **Exclude** field, type a pattern-matching string that matches files you want to exclude from the analysis. For example, temp*.* excludes all files that begin with “temp”. You can separate filters with a comma.
- 5 To select and add multiple subdirectories to a project configuration, check the **Include subdirectories** box (on by default). This causes all source files matching the filter in all subdirectories of the specified path to be added to the project.
- 6 If you want this directory to be watched for any new files or deleted files, check the **Watch this directory** box. Whenever a source file is added to or deleted from this directory, the change is reflected in this project. Watched directories are indicated by the  icon in the files list. Directories excluded from being watched are indicated by the  icon. By default, the subdirectories of a watched directory are also watched. See page 45 for watch setting overrides.
- 7 After you have set the fields, click the **OK** button to add the source files in that directory to the project. You can click **Cancel** if the add file process is taking too long.

Tip: You may add files from multiple directory trees.

If you are using Microsoft Windows, you may drag and drop a directory, a file, or a selection of files, from another window into the Project Configuration dialog to add it to the project. If you drag a folder, the Add a Project Directory dialog opens automatically. If you drag an individual file, that file will be added to the project whether it matches the file filter or not.

All directory paths are absolute.

Adding Files

To add individual source files to the project, click . You see a file selection dialog, which allows you to select one or more source files to add to the project. Browse for and select a file or files. Then click **Open**. The file(s) are added to the project.

If you click the  down arrow next to the  icon, you can choose to import a text file that contains a list of source files to import. For example, you might generate such a file from a compiler application or code management system. The file should contain one absolute file path per line. See *Adding Files to a Project* on page 300 for an example of such a file.

Removing Directories and Files

To remove a directory or file from the project, select the items you want to remove and click . The file itself is not deleted from the file system.

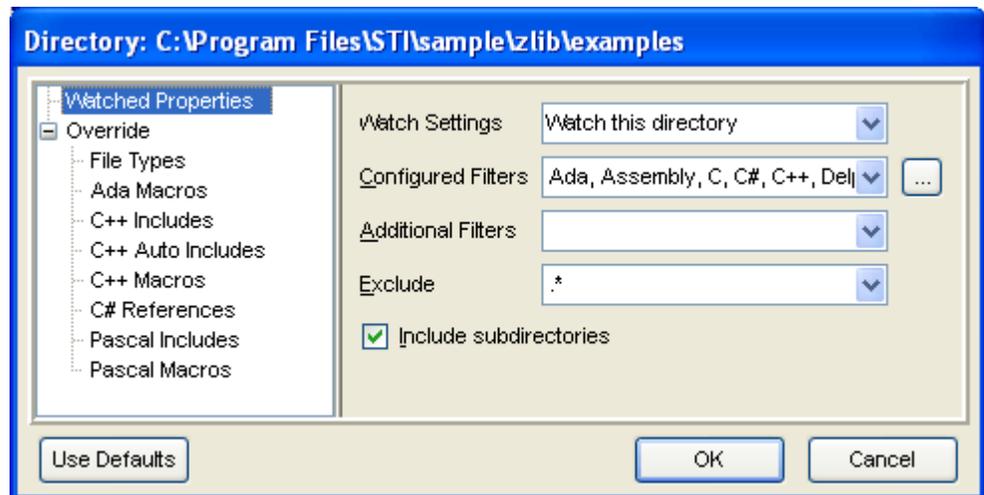
You can right-click on a removed file and choose **Add file to project** to re-add the file to the project.

Setting Overrides

Normally, each file in the project is processed according to the rules you specify in the Project Configuration window for the language of the file. For example, for C++ you can set include directories and macro definitions. However, you can override the default settings on a directory-by-directory or file-by-file basis if you like.

Directory: To override settings for a directory, follow these steps:

- 1 Select a directory.
- 2 Click  or right-click and select **Configure override settings**.

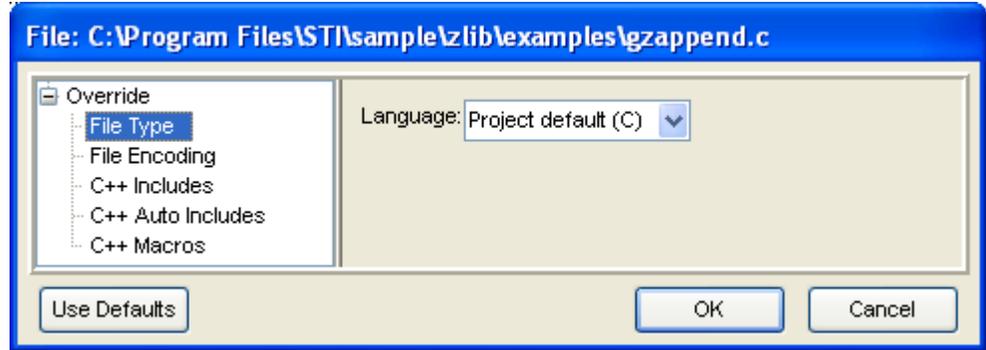


- 3 In the **Watched Properties** category, you can choose how files in this directory should be watched for new files to add to the project or deleted files to remove from the project. For **Watch Settings**, you can choose to watch a directory, not watch a directory, or inherit watch settings from the parent directory. In addition to specifying whether to watch a directory, you can set filters and exclude filters for an individual directory that control what types of new and deleted files will be found.
- 4 In the various **Override** categories, you can make directory-specific language-related settings. The list of categories depends upon the languages enabled in your project.

The **File Type** category lets you override the language of this file indicated by the file extension. The **File Encoding** category lets you override the encoding setting described in *File Options* on page 49.

File: To override settings for a file, follow these steps:

- 1 Select a file.
- 2 Click  or right-click and select **Configure override settings**.



- 3 In the various **Override** categories, select a category and make changes. The categories available are different depending on the language of the source file. See page 61 through page 89 for details. The **Watched Properties** category is available for file overrides if you are using Relative or Named Root portability.
- 4 Click **OK** to save your overrides.

Special icons in the directory tree indicate which directories are being watched , have overrides , or both .

The various **Override** categories have an **Ignore Parent Overrides** checkbox. Checking this box makes only the override settings you apply at this level (directory or file) apply; settings from higher levels are not inherited.

Scanning Watched Directories

If you set directories to be watched, you can scan those directories for new files to be added or deleted files to be removed by choosing **Project > Rescan Project Directories**.

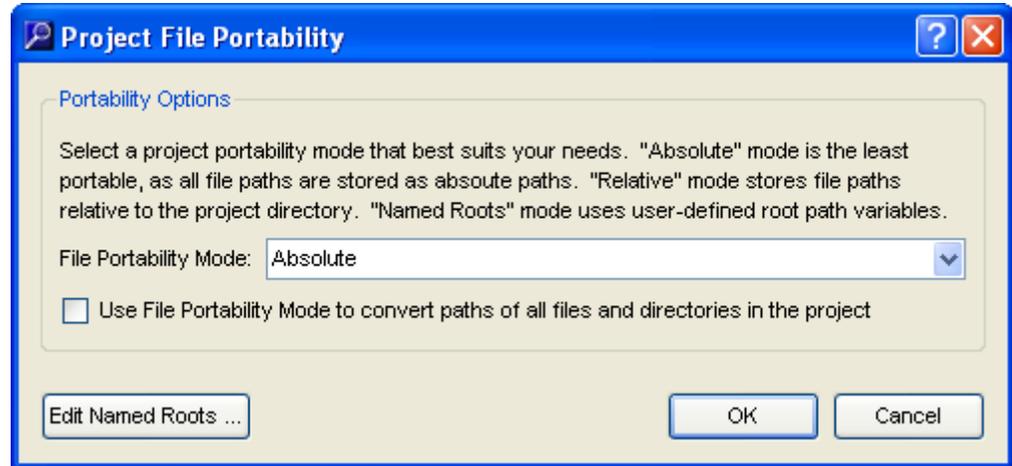
If files are found that you don't want to include in the project, uncheck the boxes next to those files to exclude them from the project configuration.

Directories that you include in the project are automatically scanned for new files when you use **Analyze All Files** to analyze the project.

You can schedule automatic scans of watched directories. See *Scheduled Activities* on page 50 for details.

Setting File Portability

You can control the portability of *Understand* projects by clicking the **Portability** button at the top of the **Files** page of the Project Configuration dialog. You will see the following dialog.



A more portable project can allow you to share the project with other users and to use the project unchanged after moving the source code files.

The choices are as follows:

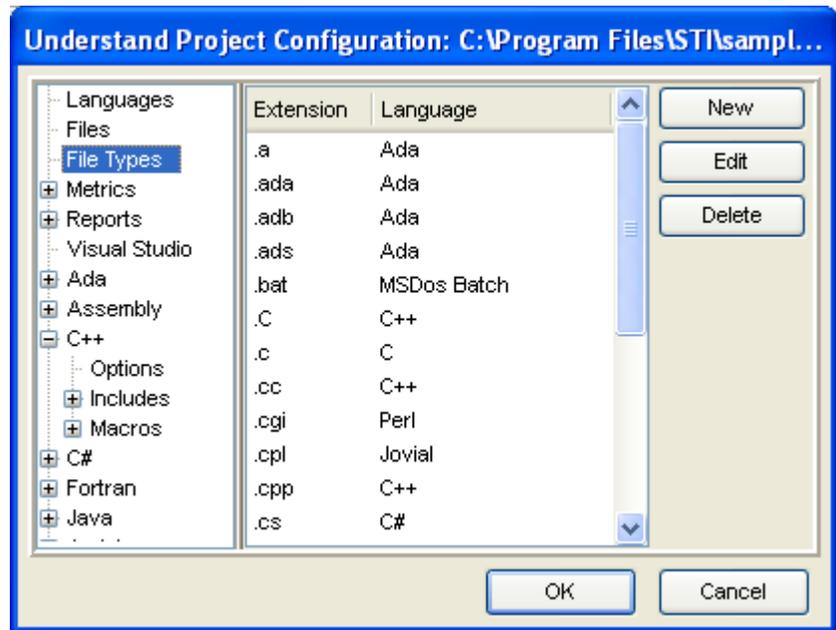
- **Absolute:** This option is the default. It stores full file paths for all directories. If the source files change location, the paths will be incorrect.
- **Relative:** This option stores the relative path to directories from the location of the *Understand* project database. If you store the project database in the source file tree and move it along with the source files, the project can still be used.
- **Named Root:** This option allows you to specify “Named Roots” that are similar to environment variables to point to a root directory. Different users may then use different definitions for a named root. Click the **Edit Named Roots** button and see page 103 for details.

Check the **Use File Portability Mode to convert paths** box if you want the file paths currently stored in the project to be updated when you click **OK**.

For more about using Named Roots, see the [SciTools Blog](#).

File Types

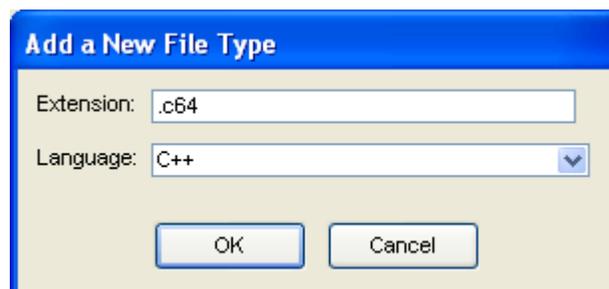
In the **File Types** category of the Project Configuration dialog, you can control how file extensions are interpreted by *Understand*.



The list shows all the file extensions already understood. Files with the types understood for the languages you checked in the Languages category are analyzed as part of the project. Other file types are not analyzed.

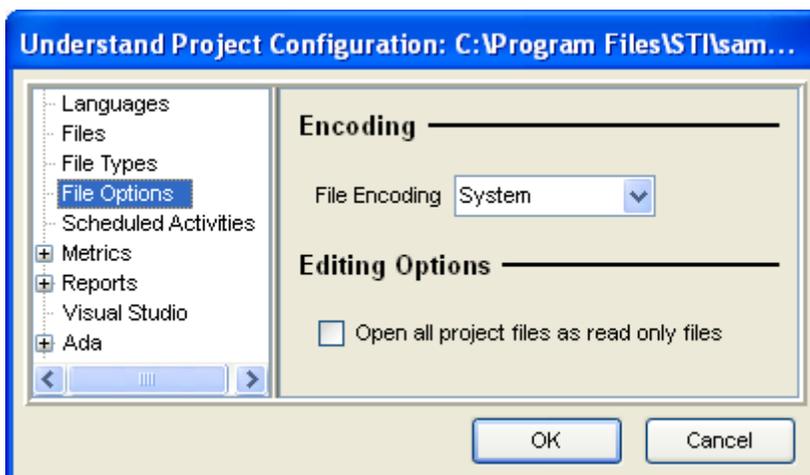
To modify an existing type, select the type and click **Edit**.

To add a file extension to the list, click **New**. Type a file extension and select the language to use for the file extension. Then click **OK**.



File Options

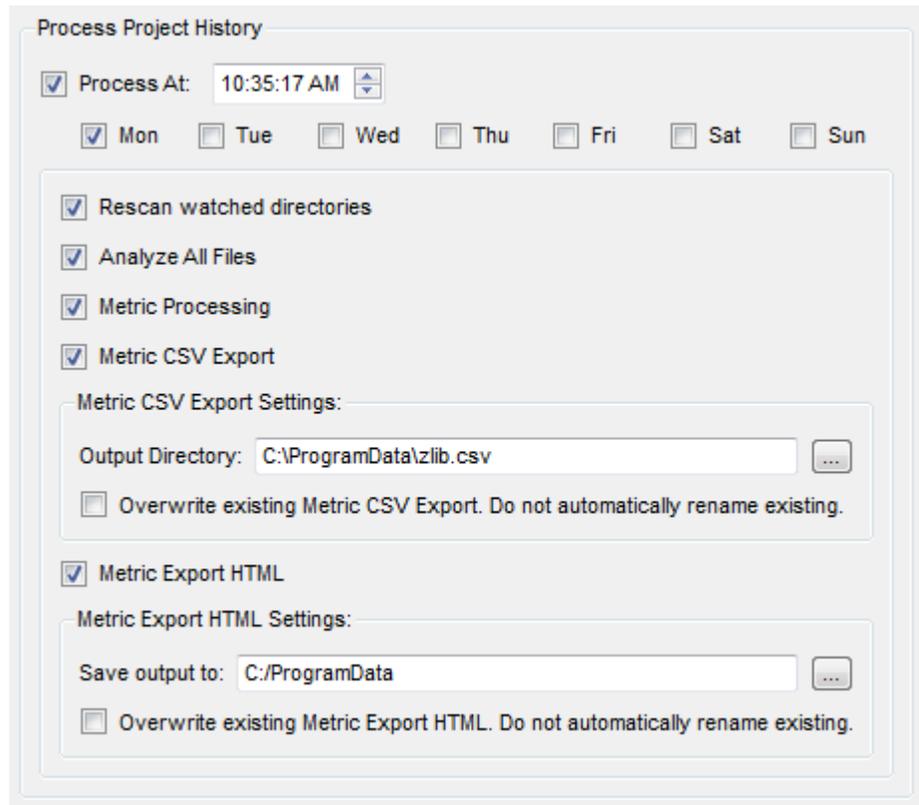
In the **File Options** category of the Project Configuration dialog, you can control how files are opened and saved by *Understand*.



- **File Encoding:** Select the type of encoding to use when saving source files. Many encoding formats are supported. You should change this only if your other applications have problems opening or displaying files saved by *Understand*. See *Editor Category* on page 105 for more information. The default file encoding is “System”, which means the default encoding for your computer. If you change the setting here, new projects you create use the last setting you saved. You can override the file encoding setting on a file-by-file or directory-by-directory basis (see *Setting Overrides* on page 45).
- **Open all project files as read only files:** Check this option if you do not want files to be edited and saved within *Understand*.

Scheduled Activities

In the **Scheduled Activities** category of the Project Configuration dialog, you can cause certain events to be performed on a regular basis. You can also open this dialog quickly by choosing **Tools > Scheduler > Scheduled Activities — <project_name>**.



To schedule events for the project you currently have open, follow these steps:

- 1 Check the **Process At** box.
- 2 Select a processing time and check the boxes for one or more days of the week.
- 3 Check the boxes for the events you want performed. The events occur in the sequence shown. For example, watched directories are scanned before the project is analyzed, and the project is analyzed before metrics are processed.

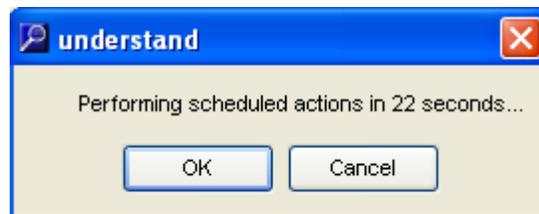
Note: *Understand* must be running at the processing time or the events will not occur.

The following activities are available for scheduling:

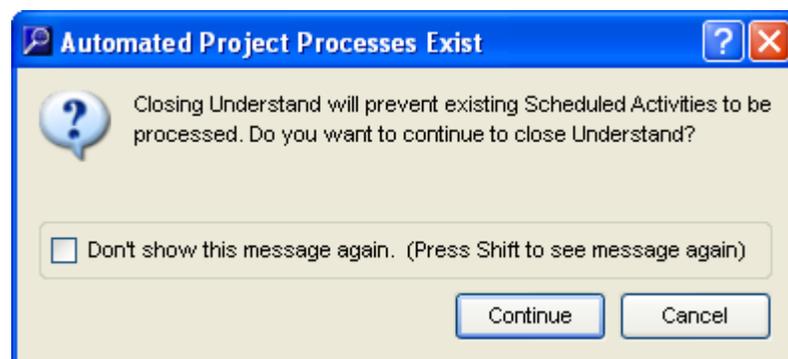
- **Rescan watched directories:** Check this box to automatically check for files that have been added to or deleted from project directories. See *Adding Directories* on page 43 for how to specify which directories to watch. If you have watched directories, you should always run this task before the “Analyze all files” task. To run this action without scheduling it, choose **Project > Rescan Project Directories**.

- **Analyze all files:** Check this box to automatically analyze all project files as described in *Analyzing the Code* on page 114. Run this task before generating any metrics so that the statistics will reflect the current state of the project. To run this action without scheduling it, choose **Project > Analyze All Files**.
- **Metric processing:** Check this box to automatically calculate metrics for the project. The metrics selected in *Metrics > Selected Category* on page 53 are processed. Run this task if you plan to schedule either of the following metrics export tasks.
- **Metric CSV export:** Check this box to automatically export metrics as a comma-separated value file. If this box is checked, you can select the directory path and output filename for the export. By default, any existing file with the same name is renamed to provide a backup. You can check the **Overwrite** box if you simply want to replace the old export file. To further configure the export, see *Metrics* on page 52. To run this action without scheduling it, choose **Metrics > Export Metrics** and see *Exporting Metrics to a CSV File* on page 218.
- **Metric export HTML:** Check this box to automatically export metrics as web pages. If this box is checked, you can select the directory path for the export. By default, any existing file with the same name is renamed to provide a backup. You can check the **Overwrite** box if you simply want to replace the old export file. To run this action without scheduling it, choose **Metrics > Project Reports** and see *Exporting Metrics to HTML* on page 217.

When scheduled activities are about to run, you see a dialog that gives you a chance to cancel the action:



If you have scheduled any activities, you see the following message when you exit from *Understand*.

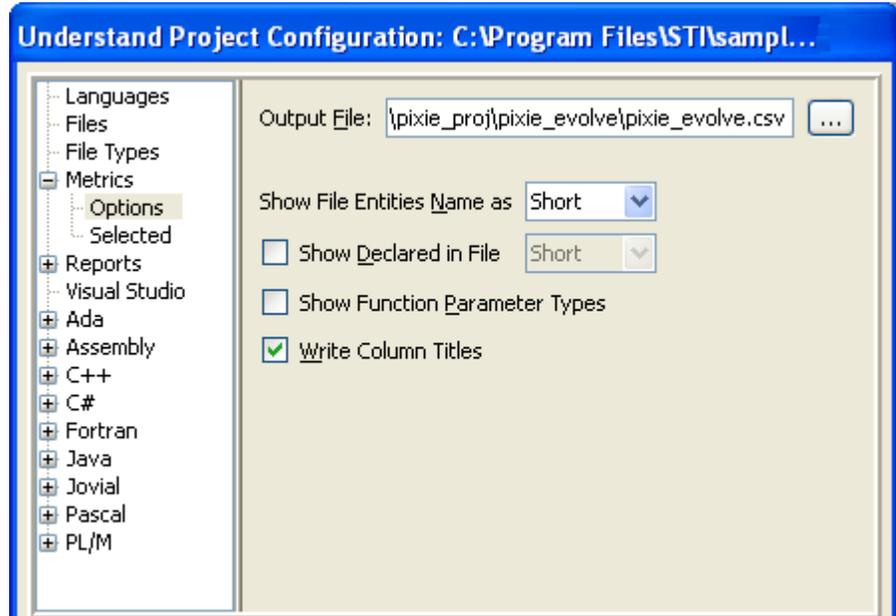


To see a list of all projects for which you have scheduled activities, choose **Tools > Scheduler > Scheduled Activities — All Projects**. To change these times, you must open the project and then use the Project Configuration dialog for that project.

Metrics

In the **Metrics** category of the Project Configuration dialog, you can control how metrics are generated when a CSV file is exported. These options set the defaults for both manual updates (page 218) and scheduled automatic updates (page 50).

The Metrics category has two sub-categories: **Options** and **Selected**.



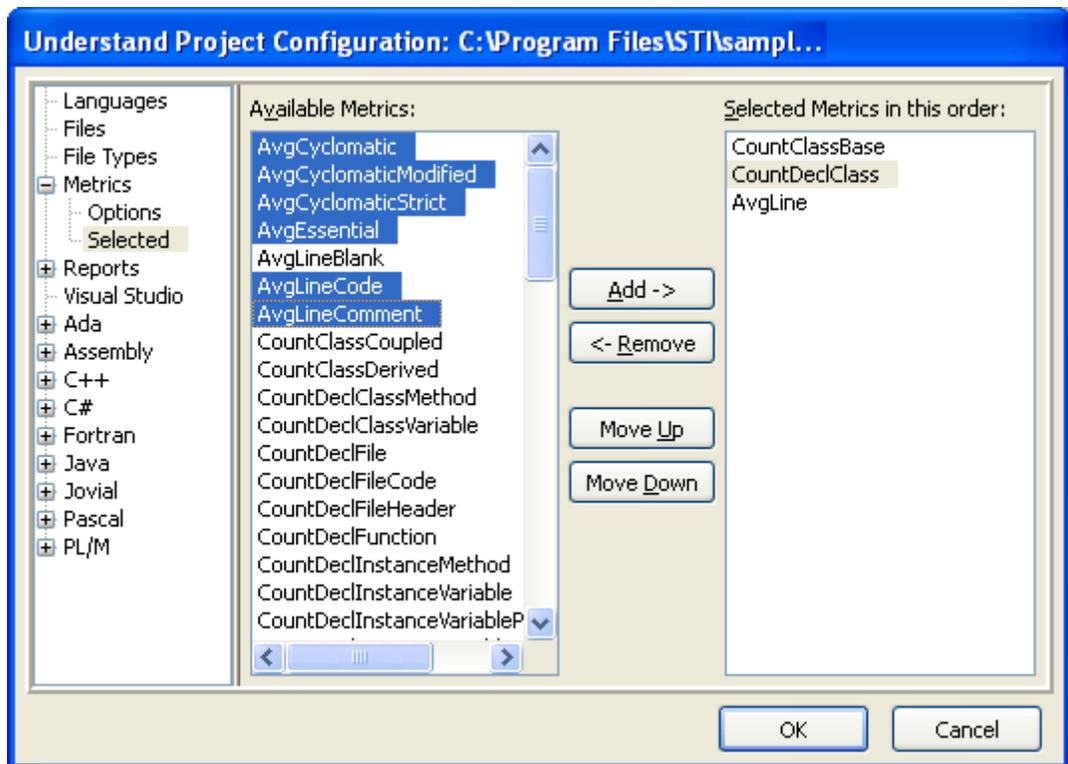
You see this window when you choose the **Project > Configure Project** menu item and then the **Metrics** category. If you attempt to generate metrics before configuring metrics, this window opens automatically.

The **Options** subcategory has the following fields:

- **Output file:** Specify the location and name of the file you want to use for metrics output. *Understand* sends its metrics output to a .csv (comma-separated values) file. This file can be opened with Microsoft Excel and other spreadsheets.
- **Show File Entities Name as:** Specify whether files should be displayed with **Short** names (just the filename), **Full** names (including the absolute path), or **Relative** names (relative directory path).
- **Show Declared in File:** Check this box if you want the file in which each entity is declared to be included in the output. You can specify whether you want these files displayed with **Short** names, **Full** names, or **Relative** names.
- **Show Function Parameter Types:** Check this box if you want the type of each function parameter listed.
- **Write Column Titles:** Check this box if you want column headings in the CSV file.

Metrics > Selected Category

The **Selected** subcategory has lists like the following:

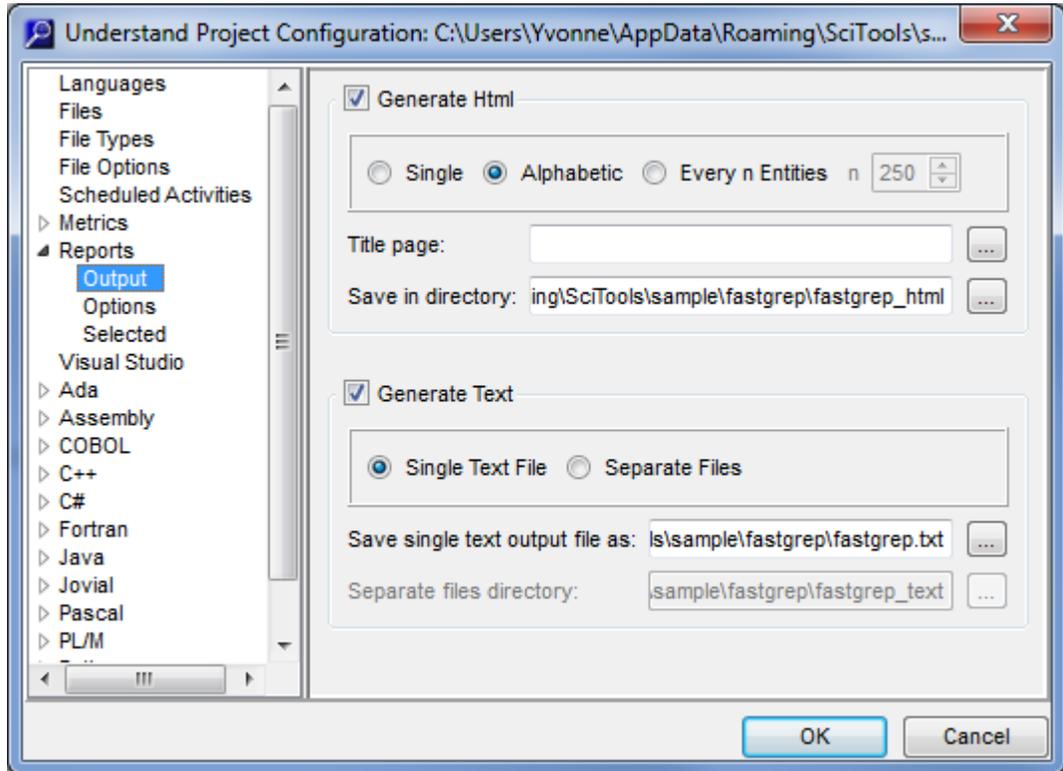


- 1 In the **Available Metrics** list (left), select metrics you want to include in the output you generate. You can hold down Shift to select a continuous group or Ctrl to select discontinuous items.
- 2 Click **Add** to copy the selected metrics to the right column.
- 3 You can reorder the metrics in the right column using the **Move Up** and **Move Down** buttons.

See www.scitools.com/documents/metrics.php for descriptions.

Reports

In the **Reports** category of the Project Configuration dialog, you can control how reports are generated. The Reports category has the following sub-categories: **Output**, **Options** and **Selected**.



This window opens if you choose the **Project > Configure Project** menu item and then the **Reports** category. You can also reach this window by clicking **Configure** in the Project Reports window.

You can control the colors and font styles in HTML reports as described in *Customizing Report Colors* on page 193.

Reports > Output Category

The **Output** subcategory has two main areas:

- **Generate HTML:** This option causes the report generation to create a large group of HTML files that are interlinked.
 - You may generate **Single** or multiple HTML files for each report type. It is recommended that you split up the files for large projects. Choose **Alphabetic** to generate multiple HTML files per report that are split up alphabetically by the first letter of the entity name. Choose **Every n Entities** to generate multiple HTML files per report that are split up every “n” number of entities. By default, a single HTML file is generated for each letter of the alphabet.

- The “home” file for the reports is index.html, but you can select an alternate **Title Page**.
- The default **Save in directory** is the <proj_file>_html folder below the folder where your .udb file is stored, but you can select an alternate location.
- **Generate Text:** This option causes the report generation to create simple text files containing the report data.
 - You may generate one text file of the specified location and name (by choosing **Single Text File**). Alternately, you may generate multiple text files (by choosing **Separate Files**) and specify a directory to contain all the files. The file extensions of each text file will denote the separate reports. Depending on which option you select, you can also select either a file or directory location for the output.

You can choose to generate either or both of the HTML and text report formats.

Reports > Options Category

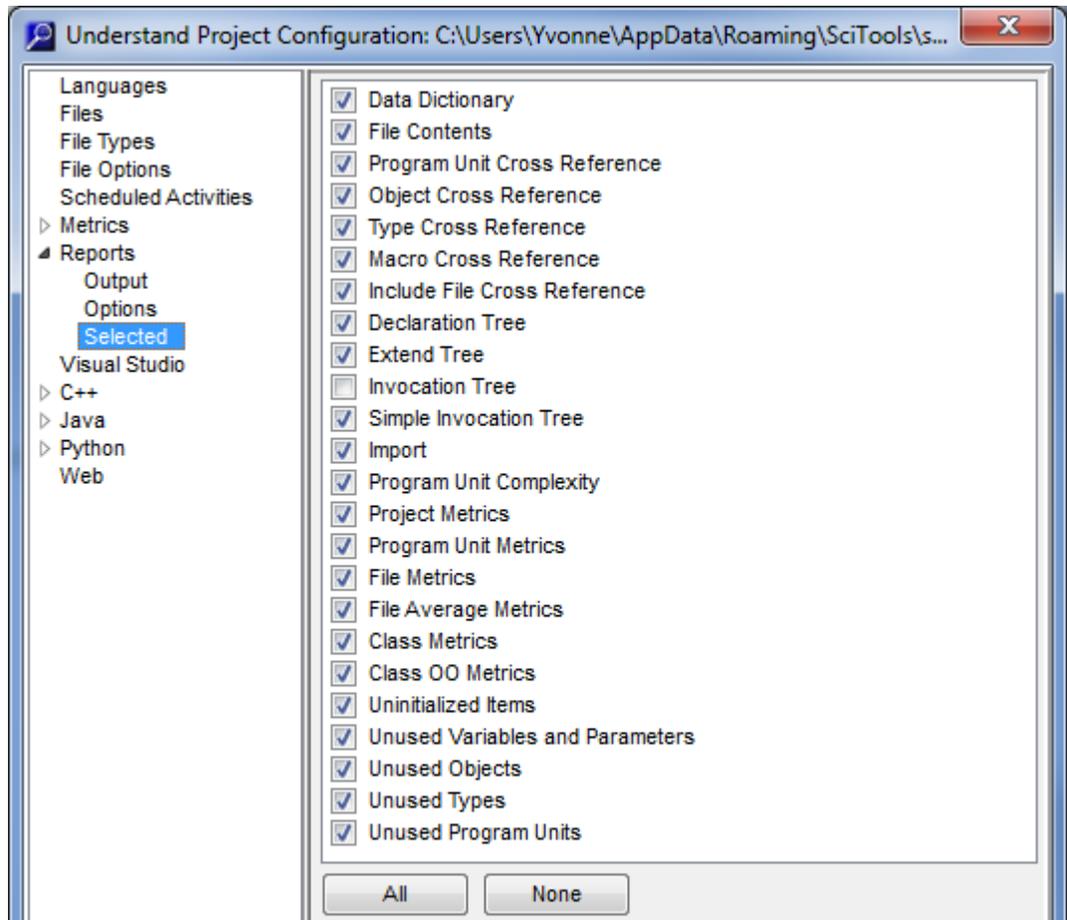
You can use the **Report > Options** category to control the contents and headers of reports.

The **Options** category has the following fields:

- **Display full filenames:** If this box is checked, the invocation tree and metrics reports show full entity names. The default is to use short names.
- **Write generation time on report:** If this box is checked, the generation date and time are included at the top of text report files. This is on by default.
- **Left aligned text:** If you check this box, the text “Report generated by Understand” will be printed in the upper-left corner of each page of the text report.
- **Right aligned text:** If you check this box, the text you provide will be printed in the upper-right corner of each page of the text report. This text can be up to 45 characters.

Reports > Selected Category

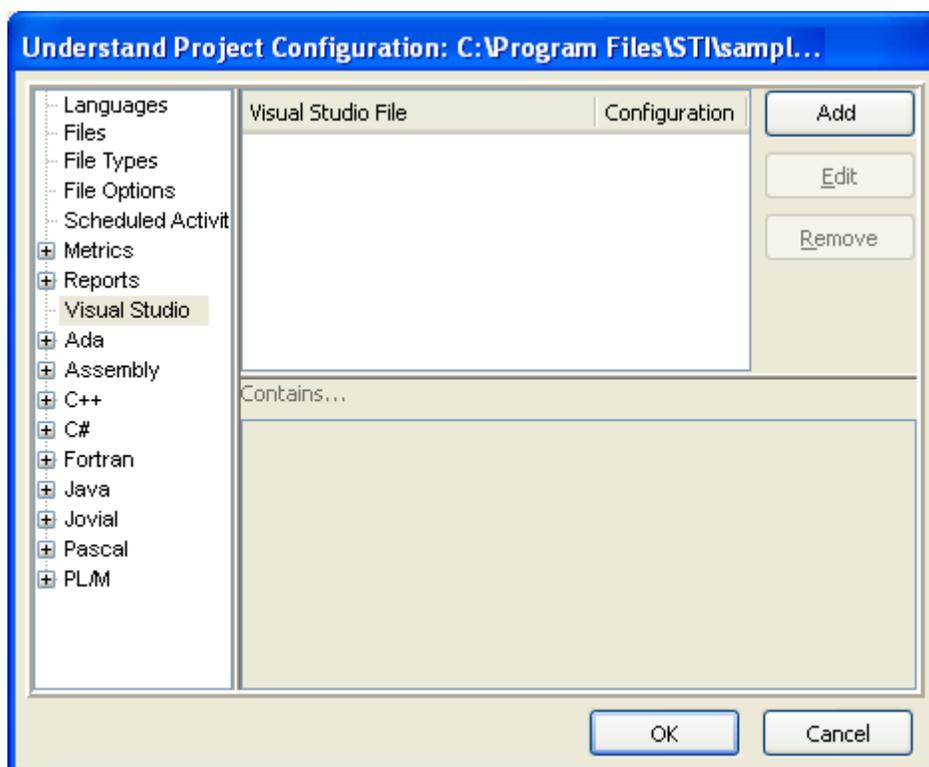
The **Selected** subcategory lets you check the boxes for the reports you want to generate. The list of reports differs depending on which languages are used in your project. See Chapter 8 for descriptions of these report formats.



Visual Studio

In the **Visual Studio** category of the Project Configuration dialog, you can tell *Understand* to use the source, macro, and include path settings from a Microsoft Studio project file.

You see this window when you choose the **Project > Configure Project** menu item and select the **Visual Studio** category.

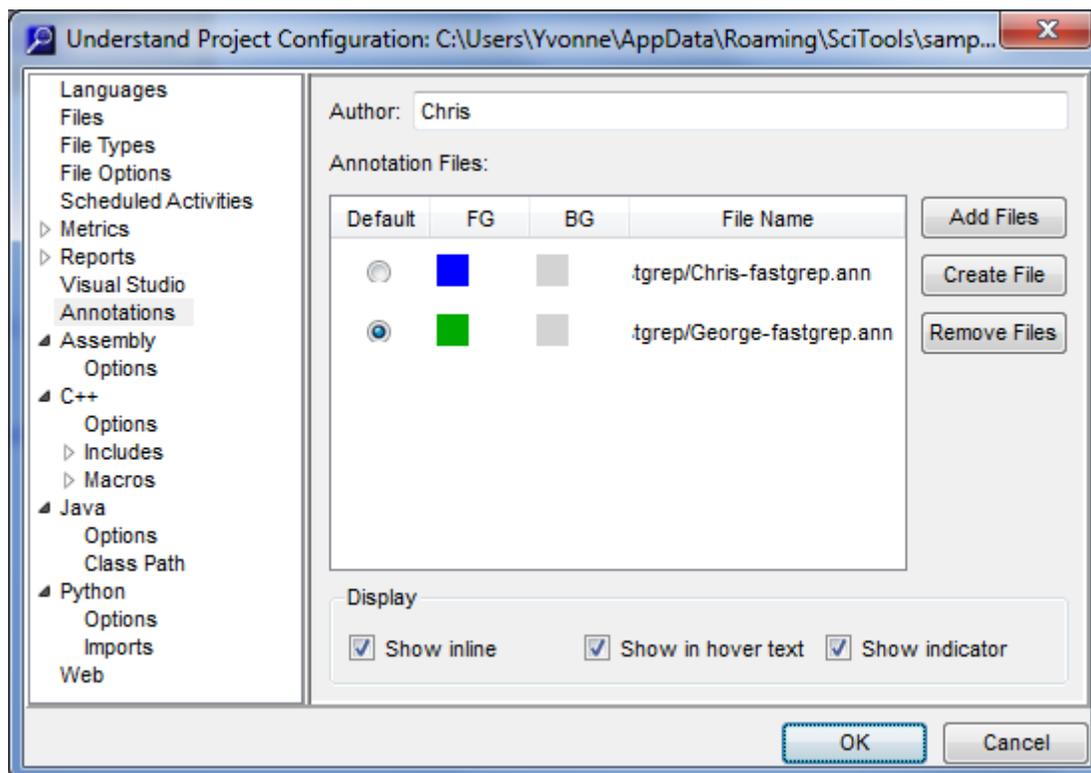


Follow these steps:

- 1 Click **Add**.
- 2 In the **Add a new Visual Studio file** dialog, click the "..." button next to **Visual Studio File**. Then browse to select a Visual Studio project file and click **Open**. MS Visual Studio project files with extensions of .csproj (C# project), .dsp, .dsw (workspace file), .sln, .vcp (Windows CE project), .vcproj (Visual C project), .vcxproj (VS2010 project), .vfproj (Visual FORTRAN project), and .vcw (workbench file) are supported.
- 3 Select the **Configuration** you want *Understand* to use when analyzing your project. You can select a project configuration or a solution configuration.
- 4 You can type an **Exclude Filter** to specify file extensions to exclude when importing a Visual Studio project.

Annotations

The **Annotations** category of the Project Configuration dialog lets you control how annotations are stored and displayed. See page 168 for details on using annotations.



- **Author:** Type your name or the username you want to be associated with the annotations you create.
- **Add Files:** Click this button to browse for an existing annotation file (*.ann). For example, you might want to add files created by other developers of this project so that you can see everyone's annotations. (If other developers are also annotating code using Understand, choose **Annotations > Refresh Annotations** from the menus when you want to get the latest annotations they have added.)
- **Create File:** Click this button to create a new annotation file. The default directory is the project directory.
- **Remove File:** Select a file and click this button to remove it from the list of files that are used to display annotations. Removing a file from this list does not delete the file from the file system.
- **Default file:** Select the file that should contain annotations you create.
- **FG:** Click on the colored block to change the text color for annotations in this file.
- **BG:** Click on the colored block to change the background color for annotations in this file.

- **Show inline:** When this box is checked, annotations are shown following the place where the associated entity is defined in the code. You can also turn this display feature on and off by choosing **Annotations > Display Inline** from the menus.
- **Show in hover text:** When this box is checked, annotations are shown if you point to a place where the associated entity is used in the code for 2 seconds. You can also turn this display feature on and off by choosing **Annotations > Display hover text** from the menus.
- **Show indicator:** When this box is checked, the entity has a squiggly line under it wherever it is used in the code. You can also turn this display feature on and off by choosing **Annotations > Display indicator** from the menus.

Annotations are stored in *.ann files, which use the SQLite database format. In addition to viewing annotations in *Understand*, you can use other applications that support SQLite to modify and search annotation files.

Ada Options

In the **Ada > Options** category of the Project Configuration dialog, you can tell *Understand* how to analyze Ada source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Ada** category.

Compiler

Version:

Preprocessor:

Standard:

Multiple Language Linkage

The case of externally linkable entities is

all lowercase all uppercase

Metrics

Count and/or operators in strict complexity

Count exception handlers in complexity

Count for-loops in complexity

Optimize

Create and cross reference record object components

Create relations between formal and actual parameters

Less memory usage versus speed

Save comments associated with entities

Options

Display entity names as:

Prompt on parse errors

Main subprograms: (main1, main2..)

Library Directories:

The fields in this category are as follows:

- **Version:** Choose the version of Ada used in your project. *Understand* supports Ada 83, Ada 95, and Ada 05.
- **Preprocessor:** Choose which type of preprocessor statements are used in your Ada code. The choices are None, C, Gnatprep, and Verdex.

- **Standard:** Choose a directory that contains a standard library used by this project. Default standards are provided within `<install_directory>/conf/understand/ada`.

Sometimes it is helpful to parse code in context of its compilation environment rather than the environment defined as “Standard” in the Ada Language Reference Manual. This is most often needed when your compiler vendor offers bindings to other languages or low level attributes of a chip or system. To do so, place all the source files containing the Ada specifications for the new standard in one directory. Then point to this directory in the **Standard** field.
- **Case of externally linkable entities:** Choose which case should be used for “exporting” entities in this language that can be linked to (for example, called as functions) by other languages. For example, if an entity is declared in this language as “MYITEM” and you choose “all lowercase” here, other languages would be expected to call that entity as “myitem”.
- **Count and/or operators in strict complexity:** Place a check in this box if you also want “and” and “or” operators considered when calculating the strict complexity metric shown in the Program Unit Complexity report. Strict complexity is like cyclomatic complexity, except that each short-circuit operator (“and then” and “or else”) adds 1 to the complexity.
- **Count exception handlers in complexity:** If this box is checked (it is on by default), exception handlers are considered when calculating the complexity metrics shown in the Information Browser and the Program Unit Complexity report.
- **Count for-loops in complexity:** Remove the check from this box if you do not want FOR-loops considered when calculating the complexity metrics shown in the Information Browser and the Program Unit Complexity report. Complexity measures the number of independent paths through a program unit.
- **Create and cross-reference record object components:** If this box is checked (off by default), separate entities are created for components of all parameters and objects of a record type. By default, all references to object components are treated as references to the record type component.
- **Create relations between formal and actual parameters:** Place a check in this box if you want the analysis to create relations between formal and actual parameters. The actual parameters linked to formal parameters include items used in expressions passed as actual parameters. This option is off by default to speed up analysis.
- **Less memory usage versus speed:** Place a check in this box if you want to use *Understand* in a very low memory consumption mode. In order to conserve memory, *Understand* frees memory used to process a program unit if that program unit is not needed. Using this option may slow down operation significantly. It is off by default.
- **Save comments associated with entities:** You can choose whether source code comments that occur before and after an entity should be associated with that entity.
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

- **Prompts on parse errors:** By default, you are prompted for how to handle errors that occur when analyzing files. When prompted, you may choose to ignore that error or all future errors. Turn this option off to disable this prompting feature. If you turned it off during analysis, but later want to turn error prompting back on, check it here.
- **Main subprograms:** Provide a comma-separated list of the names of the main subprograms in the project.
- **Library Directories:** Type a directory path or click **Edit** to browse for the location of a directory that contains Ada libraries. Library files are parsed as part of a project, but are not included in reports. All subdirectories of the directory you select will also be used to find libraries.

Ada > Macros Category

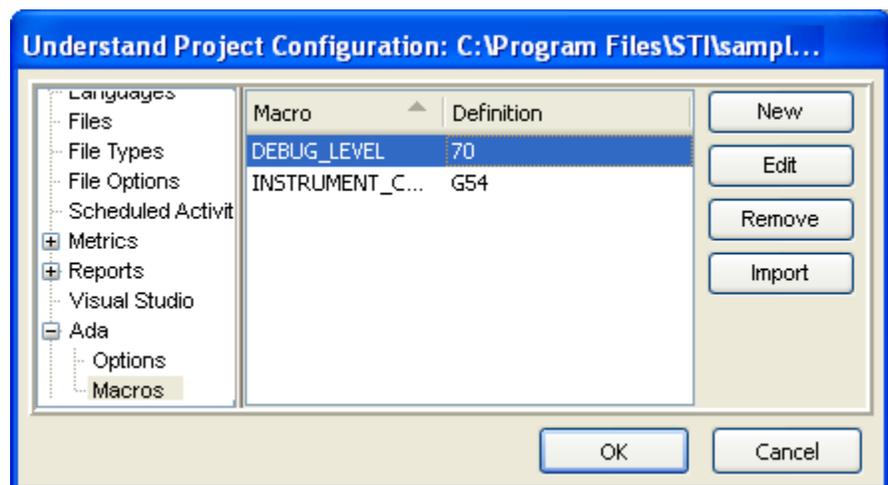
Ada code may contain conditional compiler instructions in pragma statements. For example:

```
PRAGMA IF DEVICE == D129
```

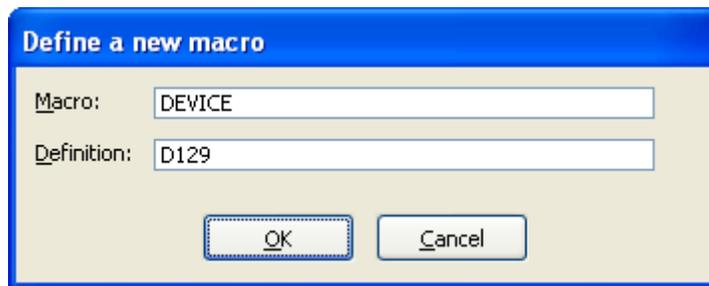
The supported pragmas are IF, IFDEF, ELSIF, ELSE, and ENDIF. These pragmas are similar to preprocessor directives such as #ifdef in C code.

For *Understand* to successfully analyze your software it needs to know what macro definitions should be set. For more about ways to configure macro definitions, see *Using the Undefined Macros Tool* on page 118 and the [SciTools Blog](#).

In the **Ada > Macros** category of the Project Configuration dialog, you can specify what macros to define for use with pragmas. You see this window when you choose the **Project > Configure Project** menu item and select the **Ada** category and the **Macros** subcategory.



The Macros category lists macros and their optional definitions. Each macro may be edited or deleted. To define a macro, click **New**.



Type the name of the macro in the first field and the definition (if any) in the second field. Then click **OK**.

A macro must have a name, but the definition is optional. Macros that have no definition value are commonly used in conjunction with PRAGMA IFDEF statements to test whether a macro is defined.

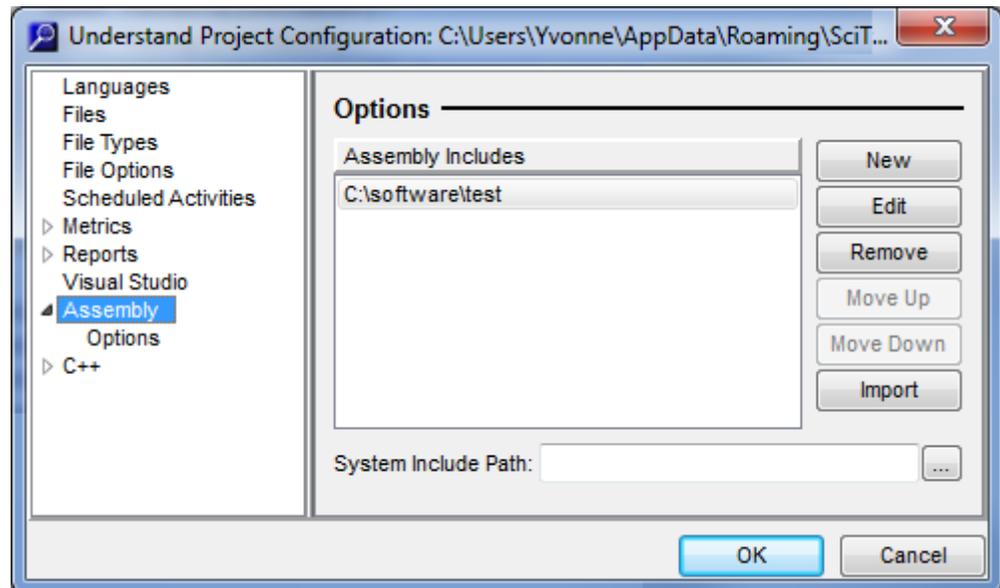
To change the definition of an existing macro without changing the name, select the macro and click **Edit**.

You can import a list of macros and their optional definitions from a text file by clicking **Import** and selecting the file. The file must contain one macro definition per line. A # sign in the first column of a line in the file indicates a comment. Separate the macro name and its definition with an equal sign (=). For example, *DEBUG=true*.

You can set macros on the und command line with the `-define name[=value]` option.

Assembly Options

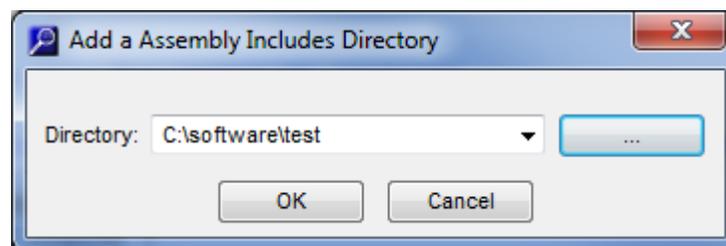
In the **Assembly > Options** category of the Project Configuration dialog, you can tell *Understand* how to analyze assembly source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Assembly** category.



You can use this category to specify include directories for assembly code. You can specify multiple directories to search for include files used in the project.

Typically only include files that are not directly related to your project, and that you do not want to analyze fully are defined here. For project-level includes that you want to be analyzed, add those include files as source files in the **Files** category.

To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.



During analysis, the include directories will be searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

For the **System Include Path**, browse to select the directory that contains system include files (include filenames surrounded by < >).

Include files found in regular include directories are added to the project. Include file found in system include directories are not added.

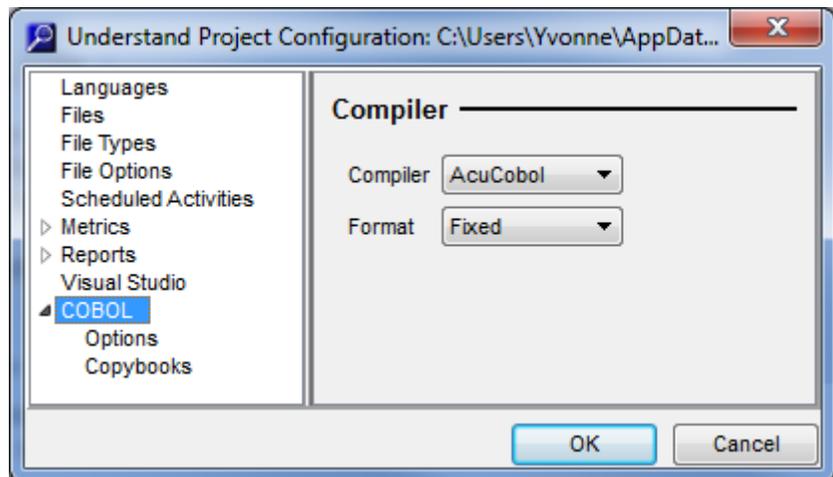
Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

You may use environment variables in include file paths. Use the `$var` format on UNIX and the `%var%` format on Windows. You can also use named root in include file paths (see page 103).

You can import a list of include directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a `#` sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

COBOL Options

In the **COBOL > Options** category of the Project Configuration dialog, you can tell *Understand* how to analyze COBOL source code. You see this window when you choose the **Project > Configure Project** menu item and select the **COBOL > Options** category.



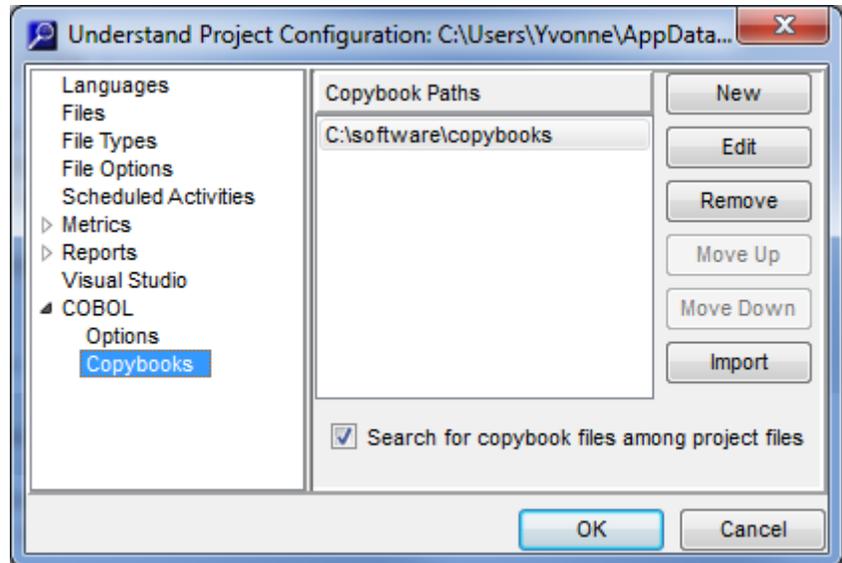
The field in the **COBOL > Options** category is as follows:

- **Compiler:** Select the compiler that you use. The options are Ansi85, MicroFocus, AcuCobol, IBM, and HP OpenVMS.
- **Format:** Choose whether the source code is in fixed or free format.

COBOL > Copybooks Category

The **COBOL > Copybooks** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify directories that contain files included with the COPY statement. Typically, such files have a *.cpy file extension. You can specify multiple directories to search for such files used in the project.

Specify directories here if they contain files that are not directly related to your project, and that you do not want to analyze fully. For copybooks that you want to be analyzed, add those files as source files in the **Files** category.



To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.

During analysis, the copybook directories are searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

If you check the **Search for copybook files among project files** box, your project directories will be searched along with any directories you specify here. When searching for a copybook, the search looks in the directories specified in this dialog first. It then searches among the project files if this box is checked.

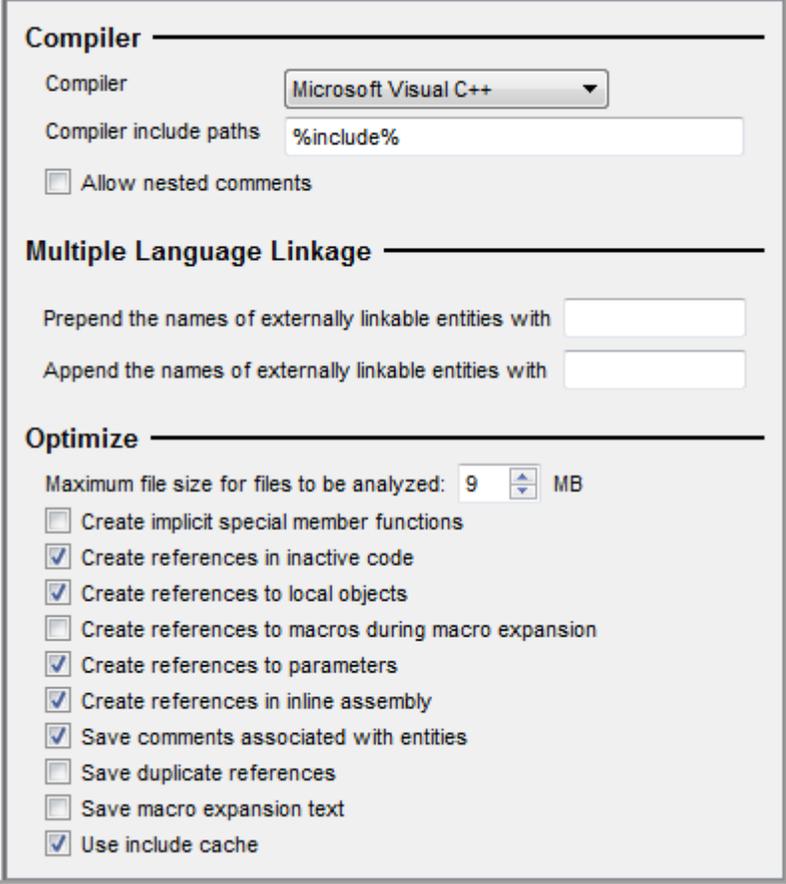
Copybook paths are not recursively searched; that is, any subdirectories will not be searched for copybook files unless that subdirectory is explicitly specified in the list of copybook directories.

You may use environment variables in copybook file paths. Use the \$var format on UNIX and the %var% format on Windows. You can also use named root in copybook file paths (see page 103).

You can import a list of copybook directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Relative paths are relative to the project file.)

C++ Options

In the **C++ > Options** category of the Project Configuration dialog, you can tell *Understand* how to analyze C and C++ source code. You see the following window when you choose the **Project > Configure Project** menu item and select the **C++** category.



Compiler

Compiler: Microsoft Visual C++

Compiler include paths: %include%

Allow nested comments

Multiple Language Linkage

Prepend the names of externally linkable entities with:

Append the names of externally linkable entities with:

Optimize

Maximum file size for files to be analyzed: 9 MB

Create implicit special member functions

Create references in inactive code

Create references to local objects

Create references to macros during macro expansion

Create references to parameters

Create references in inline assembly

Save comments associated with entities

Save duplicate references

Save macro expansion text

Use include cache

(If you selected the **Strict** option in the Languages category, see *C++ (Strict) Options* on page 76 for how to configure your project.)

The fields in the **C++ > Options** category are as follows:

- **Compiler:** Select the compiler/platform that you use. Many different compilers are supported. Your choice affects how the *Understand* parser analyzes the project. Note that not all features of a particular compiler will necessarily be handled.
- **Compiler Include Paths:** Type the path the compiler uses to find include files. For example, %include%.
- **Allow nested comments:** By default, this is off. If turned on it permits C style (`/* */`) comments to be nested. This isn't permitted by the ANSI standard, but some compilers do permit it.

- **Prepend the names of externally linkable entities with:** You may optionally type a string that you want used as a prefix to reference all linkable entities in other source code languages.
- **Append the names of externally linkable entities with:** You may optionally type a string that you want used as a suffix to reference all linkable entities in other source code languages.
- **Maximum file size for files to be analyzed:** Specify the largest file size you want to include in the analysis.
- **Create implicit special member functions:** Check this box if you want a default constructor and destructor to be created in the database and given implicit declaration references, if they are not declared in the source code for class and struct entities. This option provides entities for the parser to reference when they are called. The default is off.
- **Create references in inactive code:** If you wish to exclude cross-reference information for code that is IFDEFed out by the current macro settings, turn this option off. By default, this option is on and cross-reference information for inactive code is included.
- **Create references to local objects:** By default, all local object declarations are included in the database. If you wish to exclude variables declared within functions from the database, turn this option off. Local objects included for analysis can then be either included or excluded from the HTML output generated. Specify whether to include local objects in the HTML output on the main window of *Understand*.
- **Create references to macros during macro expansion:** Checking this box causes references to be stored during macro expansion. In some cases, this is useful. Be aware that enabling this option can add many references and make the database large and slower. The default is off.
- **Create references to parameters:** If you wish to exclude cross-reference information for parameters, turn this option off. By default, this option is on and all cross-reference information for parameters is included.
- **Create references in inline assembly:** Check this box if you want cross-references to be created to assembly code for any #asm preprocessor macros in your code.
- **Save comments associated with entities:** You can choose whether source code comments that occur before and after an entity should be associated with that entity.
- **Save duplicate references:** By default, duplicate cross-references are condensed to a single cross-reference. To keep duplicates, check this box.
- **Save macro expansion text:** If you put a check in this box, you can right-click on a macro and choose **Expanded Macro Text** from the context menu to see how the macro expands.
- **Use include cache:** By default, include files are cached during the analysis phase as they are often referenced in multiple source files. This speeds up analysis, but also uses more memory. If you have problems with excessive memory use during analysis, turn this option off. Note that there are also situations where turning the include cache on or off can affect analysis results, particularly where include actions are dependent on where they are included.

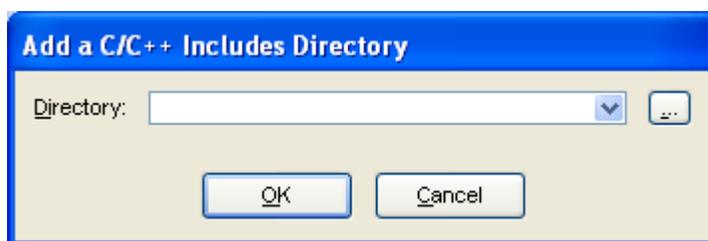
C++ > Includes Category

The **C++ > Include** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify include directories. You can specify multiple directories to search for include files used in the project.

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 116 and the [SciTools Blog](#).

Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.



During analysis, the include directories will be searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

Typically only include files that are not directly related to your project (such as system-level includes) and that you do not want to analyze fully are defined here. For project-level includes that you want to be analyzed, add those include files as source files in the **Files** category.

You may use environment variables in include file paths. Use the \$var format on UNIX and the %var% format on Windows. You can also use named root in include file paths (see page 103).

You can import a list of include directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

The **C++ > Include** category provides the following options to control include handling:

- **Add found include files to source list:** Enabling this option causes include files found during project analysis to be added to the project automatically. This allows you to see more detailed information about such include files. The default is off.
- **Add found system include files to source list:** If you choose to add include files that are found to the source list, you can also choose whether system include files should be added. The default is off.
- **Prompt for missing include files:** If any include files cannot be found during analysis, you will normally see the **Include Paths** button in the Parse Log after you analyze the project. If you want to be prompted for how to handle missing files during the analysis, you must choose **Tools > Options** and enable the **Allow**

prompting for missing include files on a per project basis checkbox in the **Analyze** category (page 101). Then, you will see this field in the Project Configuration dialog. If you then check the **Prompt for missing include files** box, you may choose to add a directory to the include path, ignore the missing file, or stop warning about missing files during the analysis.

- **Search for include files among project files:** This option directs the parser to look among project files as a last resort for missing include files. The default is on.
- **Treat system includes as user includes:** This option tells the parser to look for system includes (surrounded by < >) using the same strategies as normal includes (surrounded by quotes). If this item is off, the parser looks for system includes only in directories defined by the compiler configuration. The default is on.
- **Ignore directories in include names:** Check this option if you want to ignore any directory specifications in #include statements and instead use the include file wherever it is found in the project. The default is off.
- **Use case-insensitive lookup for includes:** This option tells the parser whether to ignore the case of filenames in #include statements. The default is off. (Not available on Windows; Windows lookups are always case-insensitive.)

C++ > Includes > Auto Category

In the **C++ > Includes > Auto** category you can specify include files that should be included before each file in a project.

To add a file, click **New** and browse for the file(s). Then click **Open**.

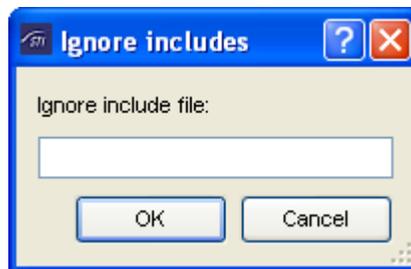
You can import a list of auto include files from a text file by clicking **Import** and selecting the text file that contains one file path per line.

Use the **Move Up** and **Move Down** buttons to change the order in which these files are included.

C++ > Includes > Ignore Category

In the **C++ > Includes > Ignore** category you can specify individual include files that you wish to ignore during analysis.

To add a file to be ignored, click the **New** button and type the filename of the include file. Then click **OK**. The filename can use wildcards, such as `moduleZ_*.h`, to match multiple files.



Any missing files you choose to ignore when prompted during analysis will be added to this list.

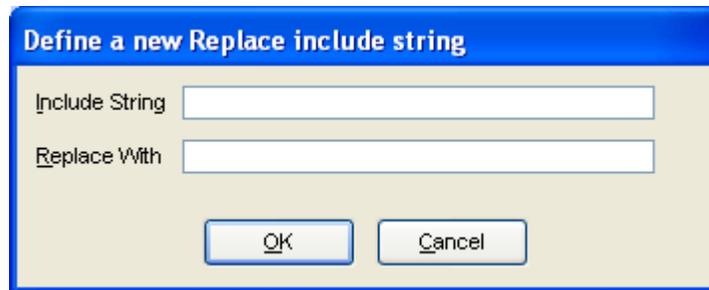
You can import a list of files to ignore from a text file by clicking **Import** and selecting the text file that contains one filename per line.

C++ > Includes > Replacement Text

In the **C++ > Includes > Replacement Text** category you can specify text that should be replaced in include file text.

For example, you might use this feature to replace VAX/VMS include paths like [sys\$somewhere] with valid UNIX or Windows paths without modifying the source code.

To add an item, type the string found in the actual include files in the **Include String** field. Type the text you want to replace it with in the **Replace With** field. Then click **OK**.



You can import a list of include strings and their replacements from a text file by clicking **Import** and selecting the file. The file must contain one include string per line. The file should separate the include string and its replacement with an equal sign (=).

Use the **Move Up** and **Move Down** buttons to change the order in which these replacements are made.

C++ > Macros Category

C source code is often sprinkled with pre-processor directives providing instructions and options to the C compiler. Directives such as the following affect what the software does and how it should be parsed:

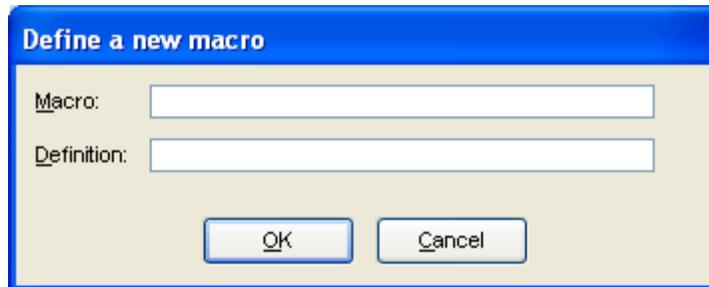
```
#define INSTRUMENT_CODE
#ifdef INSTRUMENT_CODE
... statements ...
#endif
```

Macros are often defined with directives (`#define`) in include files (.h files) or are passed in via the compiler (typically with the `-D` option).

For *Understand* to successfully analyze your software it needs to know what macro definitions should be set. For more about ways to configure macro definitions, see *Using the Undefined Macros Tool* on page 118 and the [SciTools Blog](#).

The **C++ > Macros** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to define preprocessor macros that are used when compiling the code.

To add a macro definition, click the **New** button and type the name of the macro and optionally a definition. Then click **OK**.



Note that a macro must have a name, but that the definition is optional. Macros that are defined but have no definition value are commonly used in conjunction with `#ifdef` pre-processor statements to see if macros are defined.

Note: A number of preprocessor macros are automatically supported. In additions to the common macros, *Understand* supports the following macro formats for embedded assembly code if you are using the “fuzzy” parser. (The strict C/C++ parser does not support these macro formats.)

```
#asm(<embedded assembly code>);  
#asm "<embedded assembly code>";  
#asm  
<embedded assembly code>  
#endasm
```

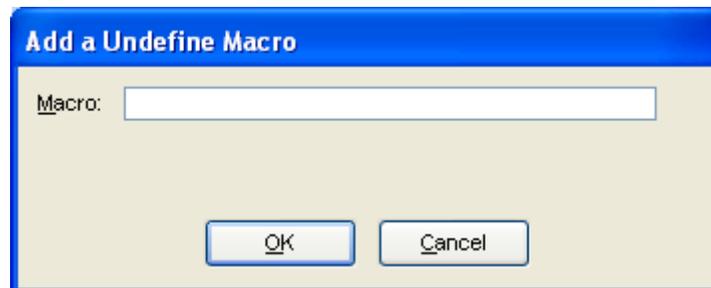
You can import a list of macros and their optional definitions from a text file by clicking **Import** and selecting the file. The file must contain one macro definition per line. A `#` sign in the first column of a line in the file indicates a comment. The file should separate the macro name and its definition with an equal sign (=). For example, `DEBUG=true`.

The priority for macro definitions is as follows, from lowest to highest priority:

- 1 Built-in language macros (`__FILE__`, etc.)
- 2 Compiler configuration file
- 3 Macro definitions in a synchronized Visual Studio project
- 4 Undefines of compiler defines (via the **Configure Undefines** button)
- 5 Project defines (Macros category)
- 6 Define on und command line using `-define`
- 7 Define in source file (`#define` / `#undef` in source)

**C++ > Macros >
Undefines Category**

You can list undefined macros in the **C++ > Macros > Undefines** category in the Project Configuration dialog. Click **New** and type the name of a macro that is not defined. Then click **OK**.



You can import a list of undefined macros from a text file by clicking **Import** and selecting the file. The file must contain one macro name per line. A # sign in the first column of a line in the file indicates a comment.

C++ (Strict) Options

See *Languages Category* on page 41 for information about the differences between the default C/C++ parser and the strict analyzer.

Note: If you did not select the **Strict** option in the Languages category next to the C/C++ box, see *C++ Options* on page 69 for how to configure your project.

In the **C++ (Strict) > Options** category of the Project Configuration dialog, you can control how C/C++ source code is analyzed. You see this window when you choose the **Project > Configure Project** menu item and select the **C++ (Strict)** category.

Target

Arch: x86_64

Vendor: pc

OS: win32

Env: unknown

Delayed Template Parsing

Language Standard

C: c89

C++: c++98

Optimize

Create references in inactive code

Save comments associated with entities

Save macro expansion text

Warnings: None Default All

The first three fields in the Target section of this dialog match target triplets used by the GNU Compiler Collection (GCC). The defaults match the platform on which you are running *Understand*. These fields are used to control which extensions (such as preprocessor defines, header search paths and language syntax) are analyzed. If your choices here do not match the code used, errors are likely to occur during the analysis.

If your code is built for multiple targets, use these options to switch between target environments for the code analysis.

The fields in the **C++ (Strict) > Options** category are as follows:

- **Arch:** Select the architecture of the chip for which your project is written. Examples include ARM, PowerPC64, and x86_64.

- **Vendor:** Select the source of the chip architecture. Options are Unknown, Apple, PC, and SCEI (Sony PlayStation). Use the “Unknown” option to select the most generic C/C++ code analysis.
- **OS:** Select the operating system that this program will be used under. Examples include iOS, Linux, and Win32.
- **Env:** Select the build environment you use to build this project. Examples include GNU, EABI, and Mach-O. For most projects, the default of “unknown” is fine.
- **Delayed Template Parsing:** If your OS is Win32, you can choose whether to delay parsing of template files. This option is required for compatibility with MSVC. However, be aware that unreferenced template code will not be analyzed at all if you enable delayed template parsing.
- **Version:** If your OS is iOS or Mac OS, you should specify the version of the operating system for which you will build the application.
- **C Language Standard:** Select the C standard to which you want your C code to conform.
- **C++ Language Standard:** Select the C++ standard to which you want your C++ code to conform.
- **Create references in inactive code:** If you wish to exclude cross-reference information for code that is IFDEFed out by the current macro settings, turn this option off. By default, this option is on and cross-reference information for inactive code is included.
- **Save comments associated with entities:** You can choose whether source code comments that occur before and after an entity should be associated with that entity.
- **Save macro expansion text:** If you put a check in this box, you can right-click on a macro and choose **Expanded Macro Text** from the context menu to see how the macro expands.
- **Warnings:** Choose how many of the warnings provided by the strict analyzer you want reported. These warnings indicate potential problems in the source code. Choosing to see some or all warnings is likely to slow down the project analysis somewhat.

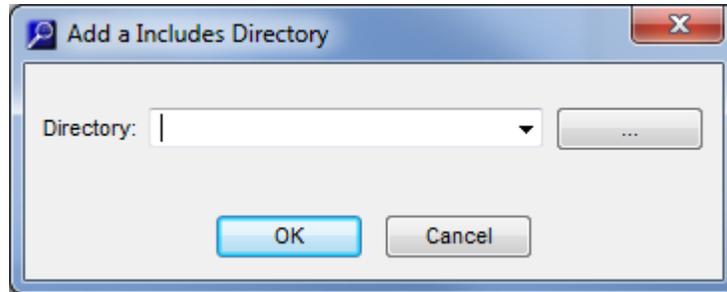
C++ (Strict) > Includes Category

The **C++ (Strict) > Includes** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify include directories. You can specify multiple directories to search for include files used in the project.

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 116 and the [SciTools Blog](#).

Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.



During analysis, the include directories will be searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

Typically only include files that are not directly related to your project (such as system-level includes) and that you do not want to analyze fully are defined here. For project-level includes that you want to be analyzed, add those include files as source files in the **Files** category.

You may use environment variables in include file paths. Use the \$var format on UNIX and the %var% format on Windows. You can also use named root in include file paths (see page 103).

You can import a list of include directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

The **C++ (Strict) > Include** category provides the following options to control how includes are handled:

- **Add found include files to source list:** Enabling this option causes include files found during project analysis to be added to the project automatically. This allows you to see more detailed information about such include files. The default is off.
- **Add found system include files to source list:** If you choose to add include files that are found to the source list, you can also choose whether system include files should be added. The default is off.
- **Search for include files among project files:** This option directs the parser to look among project files as a last resort for missing include files. The default is on.

There are a number of additional options for include file handling that are available only if you are using the default parser rather than the strict analyzer.

.....

**C++ (Strict) >
Includes >
Frameworks
Category**

In the **C++ (Strict) > Includes > Frameworks** category lets you specify Mac OS and iOS framework paths that the project uses.

To add a location, click **New** and browse for the folder. Then click **Select Folder** and then **OK**.

You can import a list of framework folders from a text file by clicking **Import** and selecting the text file that contains one path per line.

Use the **Move Up** and **Move Down** buttons to change the order in which these folders are processed.

C++ (Strict) > Includes > Prefix Headers Category

A prefix header is a C/C++ header file that is included at the beginning of every source file by the compiler. This is done without the use of a `#include` directive. It is common for Mac OS X programs to use prefix header files.

In the **C++ (Strict) > Includes > Prefix Headers** category, you can specify files that are used as prefix header files.

To add a file, click **New** and browse for the file. Then click **Open**.

You can import a list of files from a text file by clicking **Import** and selecting a text file that contains one file path per line.

Use the **Move Up** and **Move Down** buttons to change the order in which these files are processed.

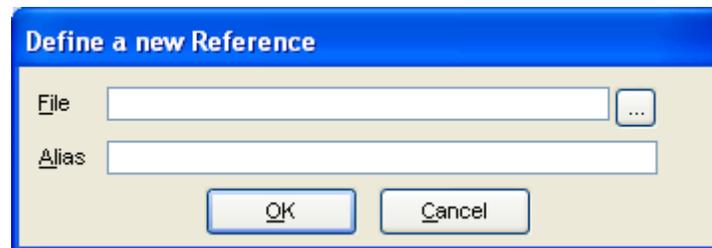
C++ (Strict) > Macros Category

For information about the **C++ (Strict) > Macros** category, see *C++ > Macros Category* on page 73.

For information about the **C++ (Strict) > Macros > Undefines** category, see *C++ > Macros > Undefines Category* on page 75.

C# Options

In the **C# > References** category, click **New**. Click **...** and browse for a .dll file. Type the alias for that file used in the code and click **OK**.



You can import a list of reference files and their aliases from a text file by clicking **Import** and selecting a file that contains one reference and its alias per line. The file should separate the reference file and its alias with an equal sign (=).

By default, reference files are analyzed as part of the project. If you do not want them to be analyzed, uncheck the **Analyze found reference files** box in this category. If this box is unchecked, methods in reference libraries are not counted for the purpose of computing metrics.

FORTRAN Options

In the **Fortran > Options** category of the Project Configuration dialog, you can specify how to analyze FORTRAN source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Fortran** category.

Compiler

Version: Fortran77

Format: Auto

Truncate column: 72 Column: 72

Allow C-style comments

Allow colons in names

Allow function declaration without parentheses

Allow quote in octal constants

Case sensitive identifiers

Use preprocessor

Intrinsic File: C:\Program Files\STI\conf\userstand\fortran\intrinsic77.txt

Multiple Language Linkage

The case of externally linkable entities is:

all lowercase all uppercase preserved

Prepend the names of externally linkable entities with:

Append the names of externally linkable entities with:

Options

Prompt on parse errors

Display entity names as: Original

The fields in the **Fortran > Options** category are as follows:

- **Version:** Select the variant of FORTRAN used by the source code in this project. If you change the version after creating a project, the project will be reanalyzed when you click OK. The choices are Fortran77, Fortran90, Fortran95, and FORTRAN2003. If you have a mix of code, choose the newest language variant. That is, if you have F77 and F95 code, choose F95. The default is Fortran95.
- **Format:** Some older FORTRAN variants and all new variants permit *free form* statements, which may cross lines). Fixed form statements are terminated by a line end or column number. The default is “auto format,” which automatically detects the parsing format (fixed or free) on a file-by-file basis. This allows you to mix free and fixed format. Auto format also determines the correct truncation point for fixed

format files. Choose “fixed” or “free” only if all your source files have the same format. Blocks of freeform code can be used within a fixed format file if you bracket the blocks with `!dec$freeform` and `!dec$nofreeform`.

- **Truncate column:** If you choose fixed form, you may choose what column terminates statements. Common columns 72 and 132 are available or you may specify a column or no truncation.
- **Allow C-style comments:** Check this option if your FORTRAN code contains comments of the form `/* ... */`.
- **Allow colons in names:** Check this box to allow colons (`:`) to be used in identifiers in F77 code. Enabling this option could cause problems in F77 code that does not use this extension, so the default is off.
- **Allow function declaration without parentheses:** Check this box if you want to allow functions to be declared without the use of parentheses. By default, parentheses are required.
- **Allow quote in octal constants:** Check this box if a double quote mark (`"`) should be treated as the start of a DEC-style octal constant. For example, `"100000`. If this box is not checked (the default), a double quote mark begins a string literal.
- **Case sensitive identifiers:** Check this box if you want identifier names to be treated case-sensitively. By default, case is ignored.
- **Use preprocessor:** Use this option to disable or enable preprocessor support.
- **Intrinsics file:** Type or browse for a file that contains intrinsic functions you want to be parsed. Default intrinsics files are provided in the `<install_directory>/conf/understand/fortran` directory: `intrinsics77.txt`, `intrinsics90.txt`, and `intrinsics95.txt`.
- **Case of externally linkable entities:** Choose which case should be used for “exporting” entities in this language that can be linked to (for example, called as functions) by other languages. For example, if an entity is declared in this language as `“MYITEM”` and you choose “all lowercase” here, other languages would be expected to call that entity as `“myitem”`.
- **Prepend the names of externally linkable entities with:** You may optionally type a string that you want used as a prefix to reference all linkable entities in other source code languages.
- **Append the names of externally linkable entities with:** You may optionally type a string that you want used as a suffix to reference all linkable entities in other source code languages.
- **Prompt on parse errors:** By default, parsing errors cause a prompt asking how to handle that error. When prompted during analysis, you may choose to ignore that error or all future errors. Turn this option off to disable this prompting feature. If you turned it off during analysis, but later want to turn error prompting back on, check it here.
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

**Fortran>Includes
Category**

The **Fortran > Includes** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify include directories. You can specify multiple directories to search for include files used in the project.

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 116 and the [SciTools Blog](#).

Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.

During analysis, the include directories will be searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

Typically only include files that are not directly related to your project (such as system-level includes) and that you do not want to analyze fully are defined here. For project-level includes you want analyzed, add those include files as source files in the **Files** category.

You can import a list of include directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

For more information, see *C++ > Includes Category* on page 71.

**Other Fortran
Categories**

For information about the **Fortran > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on page 73.

For information about the **Fortran > Macros** category, see *C++ > Macros Category* on page 73.

Java Options

In the **Java > Options** category of the Project Configuration dialog, you can specify how to analyze Java source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Java** category.

The **Java > Options** category contains the following fields:

- **Version:** Select the version of Java used by the source code in this project. If you change the version after creating a project, the project will be reanalyzed when you click OK. The choices are Java 1.3, 1.4, 5, and 6.
- **Prepend the names of JNI/KNI external entities with:** You can specify a prefix used by Java to call functions in other languages. A Java call to a function “func” would match the C function *prepend_pkg_class_func*, where *prepend* is the string you specify here, *pkg* is the Java package name, and *class* is the Java class. This follows the Java Native Interface (JNI) and the Kaffe Native Interface (KNI).
- **Include package name:** By default, the package name is included in the prefix used to call functions in other languages. Uncheck this box to remove the package name from the names of external functions.
- **Save comments associated with entities:** You can choose whether source code comments that occur before and after an entity should be associated with that entity.

Java > Class Paths Category

The **Java > Class Paths** category allows you to identify Java .jar and .class files that provide classes for which you do not have source code.

Both .jar files and .class files are supported. Jar files contain compressed .java (source) files. Class files contain compiled sources. By default, the src.jar (or src.zip) file provided by the Java Developers Kit is located. You can add other .jar files as needed.

To add a directory with .class and .java files, follow these steps:

- 1 Click **New Path**.
- 2 Locate and select the directory containing .class files. You can provide a relative path to a directory by typing the path directly in the Class Path field rather than browsing for a directory.

- 3 Click **OK**.



To add a .jar file to the list, follow these steps:

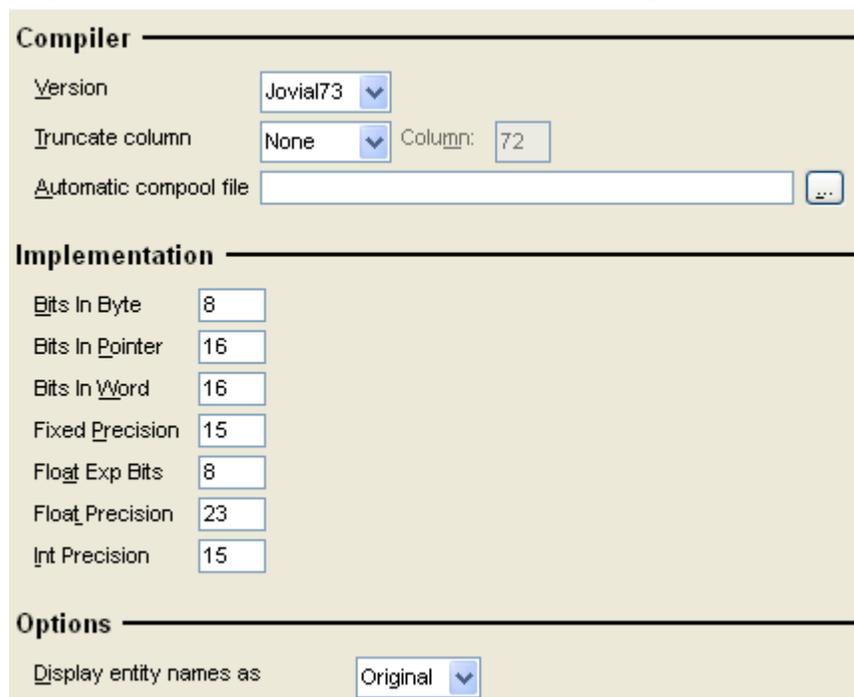
- 1 Click **New Jar**.
- 2 Locate and select the .jar or .zip file. You can select multiple .jar files while holding down the Ctrl key. You can provide a relative path to a file by typing the path directly in the Jar File field rather than browsing for a file.
- 3 Click **Open**.

If a class is found in both a .java and .class file in the class path, the class in the .java file is used.

You can import a list of class paths and/or jar files from a text file by clicking **Import** and selecting the file. The file must contain one directory or file path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

JOVIAL Options

In the **Jovial > Options** category of the Project Configuration dialog, you can specify how to analyze JOVIAL source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Jovial** category.



Compiler

Version: Jovial73

Truncate column: None Column: 72

Automatic compool file: ...

Implementation

Bits In Byte: 8

Bits In Pointer: 16

Bits In Word: 16

Fixed Precision: 15

Float Exp Bits: 8

Float Precision: 23

Int Precision: 15

Options

Display entity names as: Original

The **Jovial > Options** category contains the following fields:

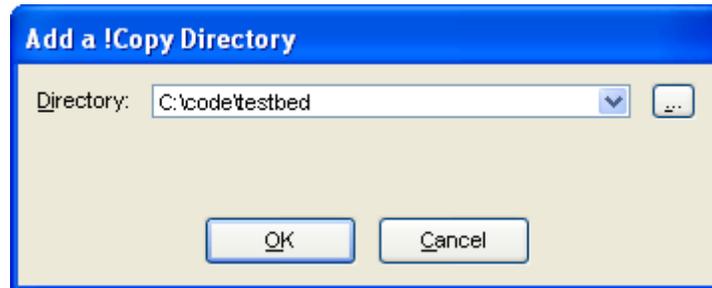
- **Version:** Select the JOVIAL version you use. JOVIAL73 and JOVIAL3 are supported.
- **Truncate column:** By default, statements are not truncated by column location. You may choose to truncate statements at column 72 or at some other user-defined column.
- **Automatic compool file:** Click ... and browse to the compool file you want to use. The file extension can be .txt, .cpl, or .jov. The selected file is automatically imported into all other files in the project.
- **Implementation fields:** The fields in this section allow you to specify the sizes and precision of various datatypes. These sizes vary with different implementations of JOVIAL. The sizes are used to determine data overlay. You can specify the number of bits in a byte, number of bits in a pointer, number of bits in a word, precision for fixed datatypes, number of bits in a floating exponent, precision for floating datatypes, and the precision for an integer.
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

**Jovial > !Copy
Category**

The **Jovial > !Copy** category in the Project Configuration dialog (which you open with **Project > Configure Project**) lets you select directories to be searched for files named in !COPY directives.

To add a directory to the list, follow these steps:

- 1 Click the **New**.



- 2 Click the ... button and browse to the directory you want to add.
- 3 Click **OK**.

When a !COPY directive is analyzed, the directories are searched in the order listed. To change the search order, select a directory and click **Move Up** or **Move Down**.

You can import a list of directories to be searched for files named in !COPY directives from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

Pascal Options

In the **Pascal > Options** category of the Project Configuration dialog, you can specify how to analyze Pascal source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Pascal** category.

Compiler

Version

Allow embedded SQL

Predeclared entities file ...

dfm converter exe ...

Multiple Language Linkage

The case of externally linkable entities is

Options

Display entity names as

The **Pascal > Options** category contains the following fields:

- **Version:** Select the version of Pascal used by the source code in this project. The choices are Compaq, Delphi, and Turbo. Select Compaq for legacy DEC Pascal projects.
- **Allow embedded SQL:** Check this box to enable parsing of embedded SQL statements in your source code.
- **Predeclared entities file:** Click ... to select a text file (*.txt) that contains predeclared routines, types, constants, and parameters used in your source code. Two versions of this file are provided in the <install_directory>/conf/understand/pascal directory: predeclared.txt and predeclareddelphi.txt. The default is set according to your choice in the Version field.
- **dfm converter exe:** Browse for and select the executable to be used to convert binary Delphi Form (DFM) files in the project to text files. The text files will then be parsed as part of the project. A number of third-party converters are available; *Understand* does not provide a converter.

- **Case of externally linkable entities:** Choose which case should be used for “exporting” entities in this language that can be linked to (for example, called as functions) by other languages. For example, if an entity is declared in this language as “MYITEM” and you choose “Lowercase” here, other languages would be expected to call that entity as “myitem”.
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

Pascal > Macros Category

The **Pascal > Macros** category allows you to add support for preprocessor macros in source code. For example, the \$IF, \$IFDEF, and \$ELSE directives are supported.

The CPU386 and MSWINDOWS macros are predefined for some types of Pascal/Delphi sources to avoid generating syntax errors with the standard library.

For more information about the **Pascal > Macros** category, see *C++ > Macros Category* on page 73.

Pascal > Namespaces Category

The **Pascal > Namespaces** category allows you to add a directory of namespaces to use when locating a unit specified in a USES statement. A USES statement may refer to a unit without specifying a namespace. So, directories you add in this category are searched in the order provided to find units with unspecified namespaces.

For example, in the following statement, Unit1 has a namespace specified, so only the namespace CompanyName.ProjectName is searched for Unit1. Since Unit2 has no namespace specified, the namespaces in the Namespaces category will be searched for Unit2.

```
uses CompanyName.ProjectName.Unit1, Unit2;
```

To add a namespace directory, follow these steps:

- 1 Click the **New** button.
- 2 Click the ... button and browse to a directory. Then click **OK**.
- 3 You can click **Move Up** or **Move Down** to change the precedence order in which the standard libraries are checked.

You can import a list of directories to use when locating units from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

Pascal > Standard Library Paths Category

The **Pascal > Standard Library Paths** category allows you to specify directories that should be searched for standard libraries.

Standard library paths are used to find units that are not found in the project files. Only files that contain the required units are processed. For example, the following statement causes the standard libraries to be searched for a unit names System:

```
Uses System;
```

The standard libraries are not used when computing project metrics.

To add a directory, follow these steps:

- 1 Click the **New** button.
- 2 Click the ... button and browse to a directory. Then click **OK**.
- 3 You can click **Move Up** or **Move Down** to change the precedence order in which the standard libraries are checked.

You can import a list of directories that should be searched for standard libraries from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line.

Pascal > Search Paths Category

The **Pascal > Search Paths** category allows you to specify directories to search for include files. To add a directory, follow these steps:

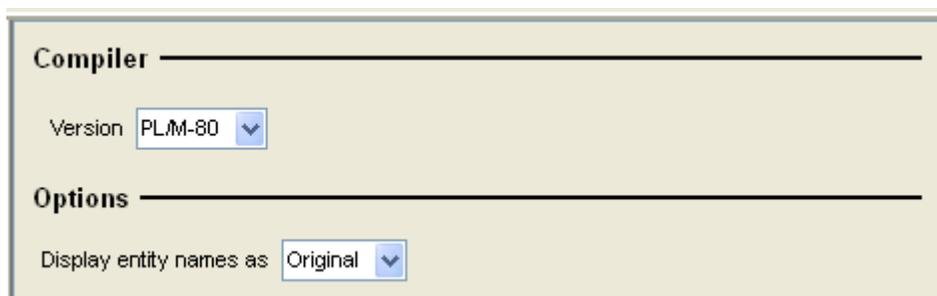
- 1 Click the **New** button.
- 2 Click the ... button and browse to a directory. Then click **OK**.
- 3 You can click **Move Up** or **Move Down** to change the precedence order in which the standard libraries are checked.

You can type a list of directory paths separated by semicolons.

You can import a list of directories to search from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line.

PL/M Options

In the **PL/M > Options** category of the Project Configuration dialog, you can specify how to analyze PL/M source code. You see this window when you choose the **Project > Configure Project** menu item and select the **PL/M** category.



The **PL/M > Options** category contains the following fields:

- **Compiler Version:** Choose the version of PL/M your compiler uses. The choices are PL/M-80 and PL/M-86.
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

.....
**PL/M>Includes
Category**

The **PL/M > Includes** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify include directories. You can specify multiple directories to search for include files used in the project.

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 116 and the [SciTools Blog](#).

Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.

During analysis, the include directories will be searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

Typically only include files that are not directly related to your project (such as system-level includes) and that you do not want to analyze fully are defined here. For project-level includes you want analyzed, add those include files as source files in the **Files** category.

You can import a list of include directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

For more information, see *C++ > Includes Category* on page 71.

For information about the **PL/M > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on page 73.

Python Options

In the **Python > Options** category of the Project Configuration dialog, you can specify how to analyze Python source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Python > Options** category.

The **Python > Options** category contains the following field:

- **Version:** Choose the version of Python you are using. The choices are Python2 and Python3.

Python > Imports Category

The **Python > Imports** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify import directories. You can specify multiple directories to search for import files used in the project.

Import paths are not recursively searched; that is, any subdirectories will not be searched for import files unless that subdirectory is explicitly specified in the list of import directories.

To add a directory, click the **New** button and then the ... button, browse to the directory, and click **OK**.

During analysis, the import directories will be searched in the order that they appear in the dialog. You can click **Move Up** or **Move Down** to change the order in which directories will be searched.

Typically only import files that are not directly related to your project and that you do not want to analyze fully are defined here. For project-level imports you want analyzed, add those files as source files in the **Files** category.

You can import a list of directories from a text file by clicking **Import** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative paths may be used. Any relative paths are relative to the project file.)

For more information, see *C++ > Includes Category* on page 71.

VHDL Options

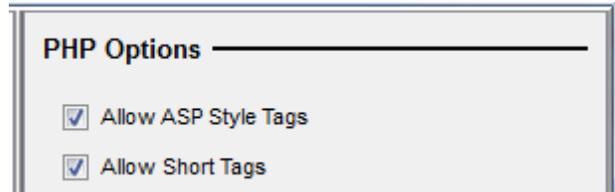
There is currently no Project Configuration category for VHDL.

If you are new to *Understand*, you should be aware that the following terms have different meanings in *Understand* than they do in VHDL:

- **Entity.** Any source construct such as a file, function, or variable. This also includes, but is not limited to, VHDL entities.
- **Architecture.** An arbitrary collection of Understand entities organized in a hierarchy. This collection may contain, but is not limited to, VHDL architectures.

Web Options

In the **Web** category of the Project Configuration dialog, you can specify what types of tags to allow in PHP files that are part of the project. You see this window when you choose the **Project > Configure Project** menu item and select the **Web** category.



Web languages included in the analysis include CSS, HTML, Javascript, PHP, and XML. For some file types, such as XML, only line count metrics are generated.

The **Web** category contains the following fields:

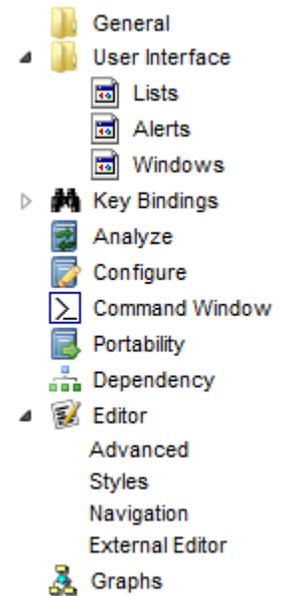
- **Allow ASP Style Tags:** Check this box if your PHP: Hypertext Preprocessor (PHP) code ever uses Active Server Pages (ASP) style tags.
- **Allow Short Tags:** Check this box if your PHP code ever uses the short form of PHP tags.

Setting General Preferences

Understand allows you to control a number of aspects of its operation using the Understand Options dialog. To open this dialog, choose **Tools > Options**. This dialog provides options to set in the categories shown to the left:

The subsections that follow describe each of the categories:

- *General Category* on page 94
- *User Interface Category* on page 96
- *User Interface > Lists Category* on page 97
- *User Interface > Alerts Category* on page 98
- *User Interface > Windows Category* on page 99
- *Key Bindings Category* on page 100
- *Analyze Category* on page 101
- *Configure Category* on page 102
- *Command Window Category* on page 102
- *Portability Category* on page 103
- *Dependency Category* on page 104
- *Editor Category* on page 105
- *Editor > Advanced Category* on page 107
- *Editor > Styles Category* on page 110
- *Editor > Navigation Category* on page 111
- *Editor > External Editor Category* on page 112
- *Graphs Category* on page 113



General Category

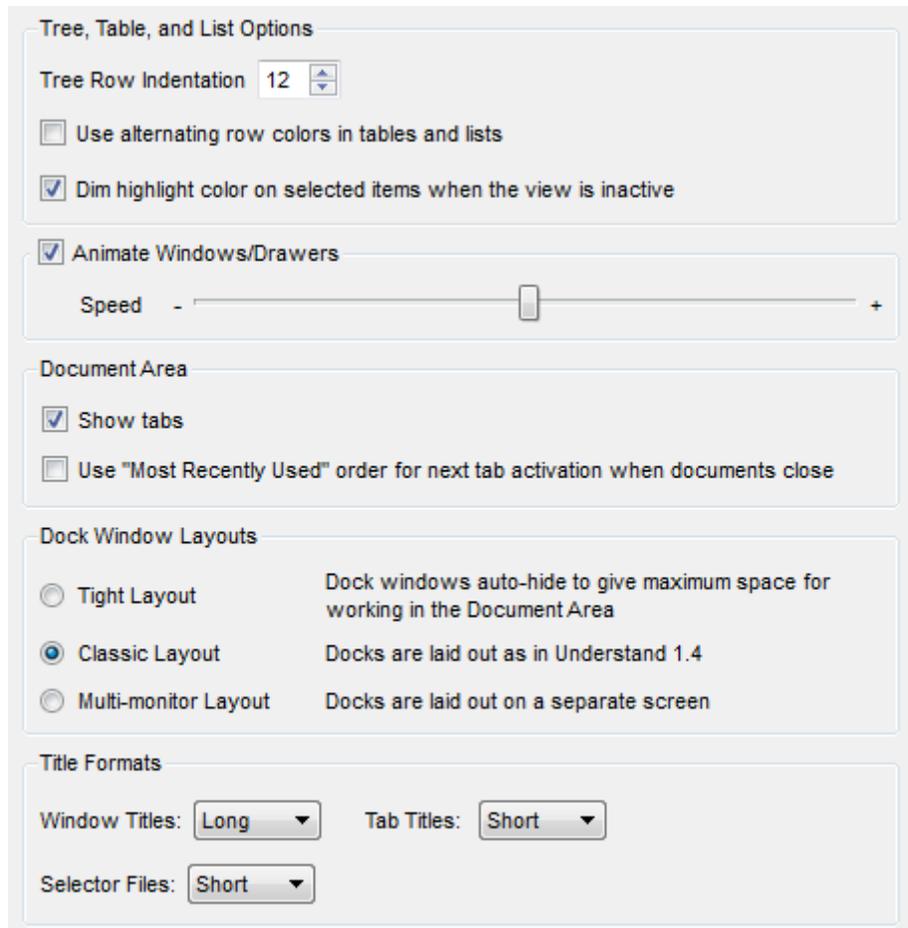
The following options can be controlled from the **General** category of the **Tools > Options** dialog:

- **Application font:** To change the font used in dialogs and lists in *Understand*, click **Change Font** and select the font, font style, and font size you want to use and click **OK**.
- **Show the Splash-Screen on startup:** If checked (on by default), the logo is shown while *Understand* is starting.
- **Show the Getting Started dialog on startup:** If checked (on by default), the Getting Started tab (see page 21) is shown in the document area when you start *Understand*.
- **Save all modified editor windows when application loses focus:** If checked (off by default), then whenever you move to another application, any editor windows in which you have made changes have their contents saved automatically.
- **Open last project on startup:** If checked (off by default), the most recently opened project is automatically opened when you start *Understand* with no other project specified. This is a useful option if you typically work with only one project.

- **Use default working directory:** If checked (off by default), you can select an alternate default directory. This will be the starting place when you are browsing for other directories and the directory to which relative directory specifications relate. The default is the directory where your project is saved.
- **Enable permissions checking for NTFS filesystems:** If you check this box, file permissions are checked on NTFS filesystems when you use the editor to modify files. This option is off by default, since this checking can significantly degrade performance in some cases.
- **Auto-detect network proxy settings:** If the Getting Started tab does not show the Scientific Toolworks blog feed, you can check this option to have your system's proxy settings checked so that the feed can be loaded. However, scanning for proxy settings takes some time and slows down the *Understand* startup process. This option is disabled by default.
- **Allow interactivity during intensive processing:** If checked (on by default), you can interact with *Understand* while it is performing background processing. Your interactive events are processed at the interval you specify in milliseconds.
- **Allow events processing every n milliseconds:** Specify how often interactive events are processed. By default, such events are processed every 100 milliseconds (0.1 seconds). You can improved background processing performance by reducing this value.
- **Application Data Location:** This field shows where files used internally by *Understand* but not associated with a specific project are stored. You can browse to change this location. You will need to restart *Understand* to have changes to this directory location take effect.

User Interface Category

The following options can be set from the **User Interface** category of the **Tools > Options** dialog:

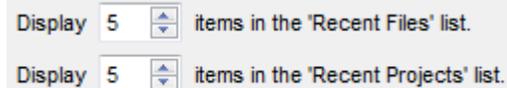


- **Tree Row Indentation:** You can change the amount of indentation in hierarchical tree displays.
- **Use alternating row colors in tables and lists:** If checked (off by default), lists and tables have shading for alternate rows.
- **Dim highlight color on selected items when the view is inactive.** The default on Windows is to dim the highlighting for the selected object when a windows loses focus. If this makes it difficult for you to read the selected object, you can change the behavior by unchecking this box.
- **Animate Windows/Drawers:** If checked (on by default), opening and closing windows and tabbed areas (drawers) is animated. You can choose a faster or slower speed than the default.
- **Show tabs:** If checked (the default), tabs are shown at the top of the document area for each of the windows open in that area. This includes the source editor windows, graphical views, and other windows.

- **Use “Most Recently Used” order for next tab activation when documents close:** If this box is checked, the most recently used window becomes the current window when you close another. If this box is unchecked (the default), the tab to the left becomes the current window.
- **Dock Window Layouts:** Choose which window layout you would like to use as the default. The **Tight Layout** is useful if you will be opening several source files and want plenty of screen space for that. The **Classic Layout** is similar to earlier versions. The **Multi-monitor Layout** allows you to take advantage of multiple screens if you have them.
- **Title Formats:** Choose whether you want filenames in the title areas of windows, tabs, and selector files to be short names, long (full path) name, or relative to the project database.

User Interface > Lists Category

The following options can be set from the **User Interface > Lists** category of the **Tools > Options** dialog:



Display 5 items in the 'Recent Files' list.
Display 5 items in the 'Recent Projects' list.

- **Recent files list:** The default is to show five items in a list of recently used files. You can change that default here.
- **Recent projects list:** The default is to show five items in a list of recently used projects. You can change that default here.

**User Interface >
Alerts Category**

The following options can be set from the **User Interface > Alerts** category of the **Tools > Options** dialog:

The screenshot shows a dialog box with five sections, each containing radio buttons or checkboxes. The sections are: 'Save On Parse' with three radio buttons (selected: Always Prompt), 'Save On Command' with three radio buttons (selected: Always Prompt), 'Project Close' with one checked checkbox (Prompt before closing the current project), 'CodeCheck' with one checked checkbox (Prompt when Violation count exceeds 300,000), and 'Entity Filter Window' with one unchecked checkbox (Sound beep when entity filter entry does not match).

These options can be used to re-enable warnings that you have disabled in a warning dialog box.

- **Save on parse:** Choose what you want done with changed but unsaved source files when the database is to be analyzed. The default is to always prompt you to choose whether to save files. Alternately, you can choose to automatically save changed files or to not save changed files.
- **Save on command:** Choose what you want done with changed but unsaved source files when a command is to be run. The default is to always prompt you to choose whether to save files. Alternately, you can choose to automatically save changed files or to not save changed files.
- **Prompt before closing the current project:** If checked (the default), you are asked whether you want to close the current project and all associated windows when you attempt to open a different project.
- **Prompt when Violation count exceeds 300,000:** If checked (the default), you are asked if you want to continue the CodeCheck when 300,000 violations have been detected.
- **Sound beep when entity filter entry does not match:** By default, the computer beeps if you type a filter in the Entity Filter that does not match any of the entities of the selected type. You can uncheck this option to turn off these beeps.

User Interface > Windows Category

The following options can be set from the **User Interface > Windows** category of the **Tools > Options** dialog:

You can choose whether to open source code files and graphical views in multiple document interface (MDI) or single document interface (SDI) windows. MDI windows are all contained within the document area of the main *Understand* window. SDI windows are separate windows that you can move anywhere on your desktop. The default is MDI.

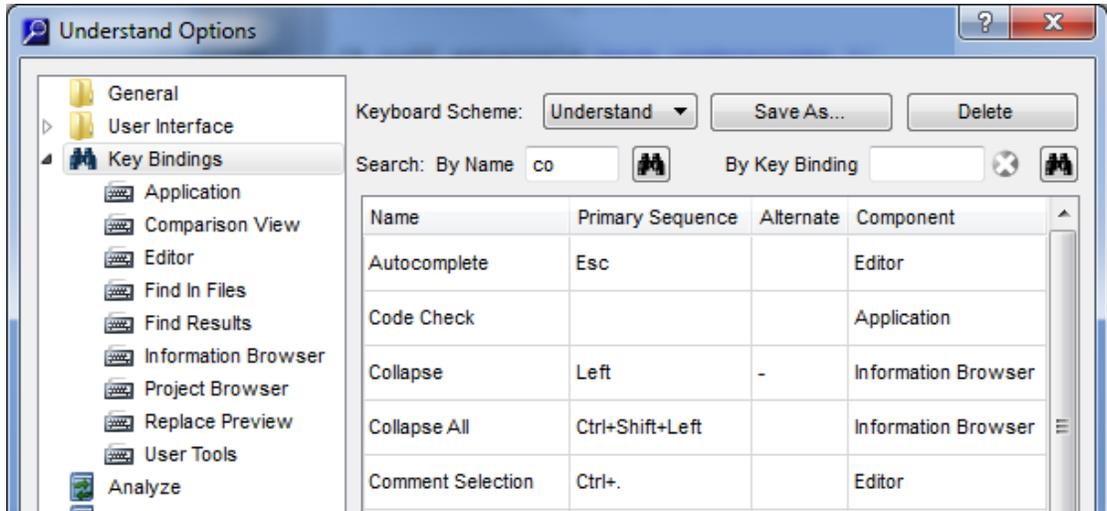
For graphical views, you can also choose to reuse a window when you open the same type of graph, but for a different entity. The default is to open a separate window for each graphical view.

- **Reuse window when requesting a similar type of graph from within window:** If checked (on by default), new graphs open in the same window from which they were opened. This occurs if you choose to view a similar graph for an item listed in a graph window.
- **Find in Files Result Window:** Click **Restore Default Position** if you want to redock the results for the Find in Files dialog to the bottom of the *Understand* window.
- **Selection Color:** Click the colored square to choose a different color for the selected result in the Find in Files and CodeCheck results.
- **Expand/Collapse code snippet when result double-clicked:** By default, code found in the Find in Files results is expanded to show surrounding lines of code when you double-click. Uncheck this box if you don't want this behavior to occur.

- **Visit result in editor when result double-clicked:** If you check this box, the code is shown in the Source Editor when you double-click on a result.
- **Use alternating row colors:** By default, the results for Find in Files and CodeCheck have a slightly darker background for every second row. You can turn off this shading by unchecking this box.

Key Bindings Category

The functions of keys in *Understand* can be customized. The **Key Bindings** category of the **Tools > Options** dialog lets you choose how keys will work in *Understand*:



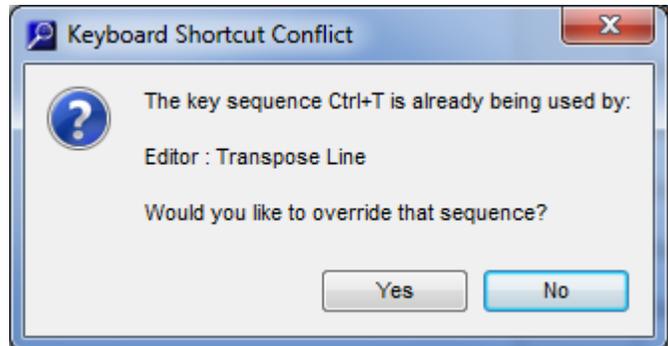
- **Keyboard Scheme:** This field allows you to choose groups of keyboard settings that are similar to other applications. The default settings are those native to *Understand*. Other choices are Visual Studio .NET key bindings and the Emacs editor key bindings. If you choose a scheme and click **OK**, that scheme will be used. If you make a change to one of the provided schemes, that becomes a “Custom” scheme. You can click **Save As** to name and save your key binding scheme.
- **Search By Name:** Type part of a command name and click the Find icon. All commands that contain that string will be shown.
- **Search By Key Binding:** Click on the field and press the key sequence you want to search for. Then click the Find icon. for example, press F3 to find all the key bindings that contain the F3 key.
- **Component:** Different portions of *Understand* have different key behaviors. The “Component” column in the table indicates where a particular command is available. You can see the key bindings for a particular component by selecting a sub-category under the main Key Bindings category in the left side of the dialog. (The Application component applies to dialogs and items not otherwise listed.)

To see a full list of all the current key bindings, choose **Help > Key Bindings**.

To change the key sequence for an action, follow these steps:

- 1 Use the Component categories or the Search fields to find a command whose key binding you want to change.

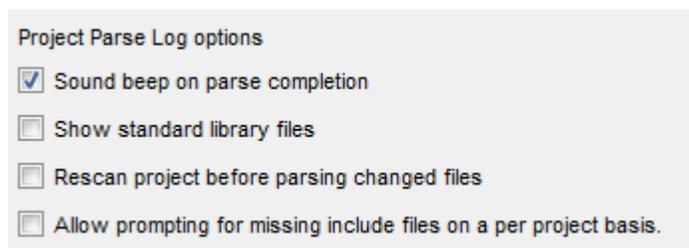
- 2 Put your cursor in the **Primary Sequence** or **Alternate** column for the command you want to modify.
- 3 Press the key combination you want to use to perform that action.
- 4 You can't use normal editing keys like Backspace or Delete to edit the keys shown in these fields. To delete the key combination you have entered, click the **X** in the red circle.
- 5 When you move focus away from a key binding you changed, you may see a warning message if the key combination you chose is already used. For example:



- 6 Click **Yes** to make the change or **No** to cancel the change. Use the **Restore Defaults** or **Cancel** button if you make changes you don't want to save. Or, you can choose one of the provided **Keyboard Schemes** to go back to a default set of key bindings.

Analyze Category

The **Analyze** category of the **Tools > Options** dialog allows you to specify options for how the project is analyzed.



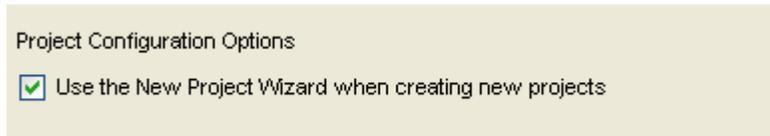
- **Sound beep on parse completion:** By default, a beep notifies you when the analysis is complete.
- **Show standard library files:** For languages whose standard libraries are parsed by *Understand* (such as Ada), if you check this box the standard library files are shown in the parse log. By default, this box is not checked, and the parse log is shorter.
- **Rescan project before parsing changed files:** If you check this box, *Understand* scans for files that have been added to project directories and to any Visual Studio projects referenced by this *Understand* project before analyzing the files currently in

the project. This has the same effect as using the **Project > Rescan Project Directories** menu command before analyzing the project. By default, this option is off.

- **Allow prompting for missing include files on a per project basis:** If you check this box, the **Prompt for missing include files** box becomes visible in the C++ > Includes category of the Project Configuration dialog. See *C++ > Includes Category* on page 71.

Configure Category

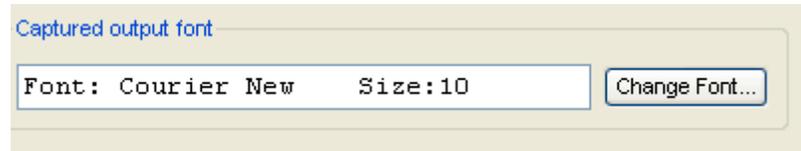
The following options can be set from the **Configure** category of the **Tools > Options** dialog:



- **Use the New Project Wizard when creating new projects:** The check in this box causes the New Project Wizard (page 35) to be used when you choose **File > New > Project**. If you uncheck this box, you can specify a project database location and filename and then use the full Project Configuration dialog.

Command Window Category

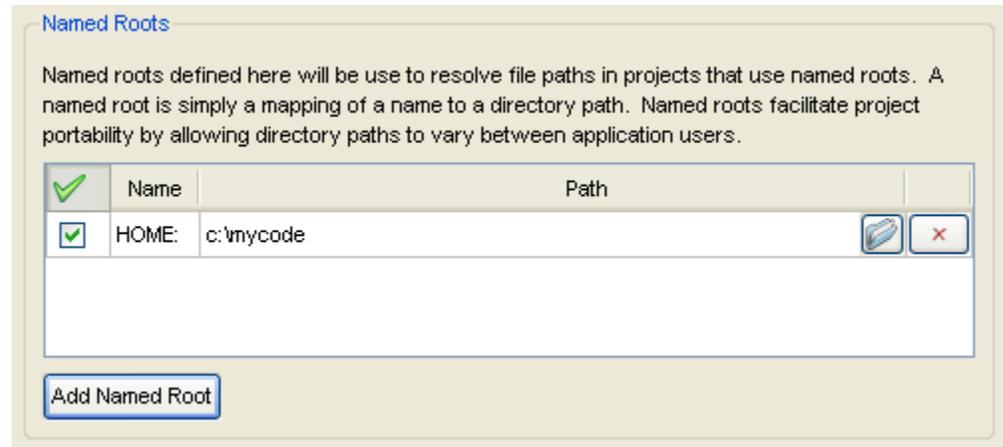
The following option can be set from the **Command Window** category of the **Tools > Options** dialog:



This setting controls the font used in the Run a Command dialog to display output from the commands you issue.

Portability Category

The Portability category of the **Tools > Options** dialog lets you specify names to use as substitutes for file paths. Named roots are similar to environment variables.



After you have defined a named root, you can use that name in other *Understand* dialogs, such as the Project Configuration, and in “und” command lines (see page 298). This is useful, for example, if you want to share projects with people who reference project files over a network using different paths.

To add a named root, click the **Add Named Root** button. This adds a new row where you can type a name and a path (or click the folder icon to browse for the location).

You can uncheck one or more named roots if you want to temporarily deactivate certain names.

If you change a named root, the project will most likely need to be re-analyzed.

You can define operating system environment variables that will be used as named roots in *Understand*. At the operating system level, define environment variable that have a prefix of “UND_NAMED_ROOT_”. The prefix is not used when you reference a named root within *Understand*. For example, suppose you define a system environment variable as follows:

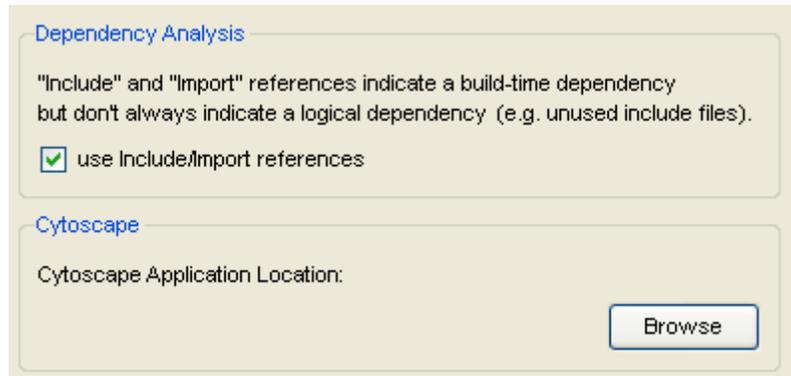
```
UND_NAMED_ROOT_SOURCEDIR=c:\my\project\dir
```

The named root the you use within *Understand* is “SOURCEDIR”.

To use a named root, see *Setting File Portability* on page 47.

Dependency Category

The Dependency category of the **Tools > Options** dialog lets you set options related to the Dependency Browser, dependency graphs, and dependency exports.



- **Use Include/Import References.** By default, “includes” and “imports” are treated as dependencies. However, you may want to omit such relationships from dependency lists if they are required for building but are not logically dependent.
- **Cytoscape Application Location.** You can browse for the location where you installed Cytoscape (www.cytoscape.org), a free open-source program for analysis and visualization. Specifying this location allows *Understand* to open Cytoscape for viewing the dependency XML files exported as described in *Exporting Dependencies to Cytoscape* on page 227.

Editor Category

The following options can be set from the **Editor** category of the **Tools > Options** dialog:

The screenshot shows the 'Editor' category settings in the 'Tools > Options' dialog. The settings are organized into several sections:

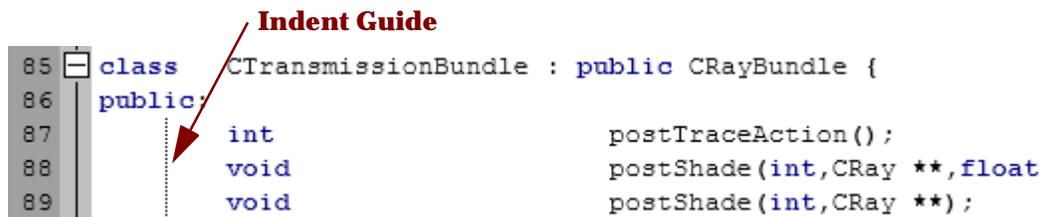
- Default Style:** Font: Courier New, Size: 10, Antialias checked.
- File Mode:** Encoding: System, Line Endings: CRLF (Windows), On Save: Convert existing line endings, Convert tabs to spaces, Add newline at end of file if absent (checked), Remove trailing whitespace.
- Caret Line:** Highlight Caret Line checked, Color: white.
- Externally Modified Files:** Always Prompt selected, Automatically Reload, Automatically Ignore.
- Indent:** Show Indent Guide, Insert Spaces Instead of Tabs, Indent Width: 4, Tab Width: 4.
- Page Guide:** Show Page Guide, Column: 80.
- Whitespace:** Invisible selected, Always Visible, Visible After Indent, Show End-of-Line.
- Margins:** Line Number, Bookmark, and Fold checked.

- **Default style:** Use the **Font** pull-down list to select a font for Source Editor windows. The fonts shown are the fixed-width fonts available on your system. Select a **Size** for the Source Editor text. If you check the **Antialias** box, the font is smoothed. The fields in this area set the default size. You can change it on a per-file basis by choosing one of the **View > Zoom** menu options.
- **File Mode:** Select the type of **Encoding** to use when saving source files and the **Line Endings** character you want used. Many encoding formats are supported. The “System” encoding uses the same encoding format defined for your operating system. You should change these settings only if your other applications have problems opening or displaying files created by *Understand*. By default, these settings apply only to new files you create, including text and CSV files. The previous format is preserved for existing files. However, if you check the **Convert existing line endings** box, files you save are converted to the format chosen here.

- **Windows** line-endings are terminated with a combination of a carriage return (`\r`) and a newline (`\n`), also called CR/LF.
- **UNIX** line-endings are terminated with a newline (`\n`), also referred to as a linefeed (LF).
- **Classic Macintosh** line-endings are terminated with a single carriage return (CR).

If you check the **Convert tabs to spaces** box, tabs are changed to the number of spaces specified in the **Width** field when you save the file. Also, if you check the **Add newline at end of file if absent** box, a new line character is added to a file that doesn't have one when you save the file (checked by default). If you check the **Remove trailing whitespace** box, any spaces or tabs at the end of lines is deleted automatically when a file is saved.

- **Caret Line:** Check the **Highlight Caret Line** box if you want the full line on which your cursor is located to be highlighted. The default **Color** is light gray, but you can change that by clicking the color box and using the Select Color dialog.
- **Externally Modified Files:** If an open file is changed in some other program, *Understand* detects this. Choose **Always Prompt** if you want to be notified and asked to load that changed version. **Automatically Reload** does this without prompting. **Automatically Ignore** is dangerous and not recommended.
- **Indent:** Check the **Show Indent Guide** box if you want a dotted line to show to column to which lines should be indented.



By default, the **Insert Spaces Instead of Tabs** box is off; turning it on adds spaces to a source file when you press `<Tab>`.

For **Indent Width**, specify the number of columns in an indentation level. For **Tab Width**, specify the number of columns for each tab stop. For example, if you set the Tab Width to 4, each `<Tab>` moves 4 columns to the right. If you set Indent Width to 6 and Tab Width to 4, each automatic indentation level is made up of one `<Tab>` and 2 spaces. See *Editor > Advanced Category* on page 107 for advanced indentation options.

- **Show Page Guide:** Check the **Page Guide** box to display a line similar to the Indent Guide at a defined line width (that is, at the right edge of the code). Set the **Column** to the character width you want to see indicated.
- **Whitespace:** Select whether you want to see indicators about whitespace characters. A dot indicates a space, and an arrow indicates a tab. You can choose Invisible (the default), Always Visible, or Visible after Indent. Check the **Show End-of-Line** box to see the characters that force a line break.

- **Margins:** Check **Line Number** (on by default) to turn on line numbering in the source view. Check **Bookmark** (on by default) if you want bookmarks (red arrows) shown in the margin next to line numbers. Check **Fold** (on by default) to turn on the ability to “fold” source code entity blocks out of the way.

```

5  ▶ #define __nested__
6  #define __config__
7  □ #ifndef std_include
8  ⊕ #ifndef nested
15  L #endif

```

Editor > Advanced Category

You can further customize the code editor’s behavior in the Options dialog. To open this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Advanced** category.

The following options control how source code looks when you print it from an editor window:

- **Font Size:** Choose the size of the source code you want to use for printing. To zoom in and out in an individual source code window, see page 167.
- **Color Mode:** Choose a color mode for printing. The choices are as follows. Note that colors other than black and white are printed only if you are using a color printer and the printer driver is set to print in color.
 - “Normal” matches the current display appearance.
 - “Invert Light” prints black as white and white as black. This is useful if you set the background to a dark color and the text to light colors for your display.
 - “Black on White” prints black code on a white background regardless of the current display appearance.

Print

Font Size: 10

Color Mode: Normal

Wrap Mode: Wrap Word

Copy-and-paste

Include line numbers in rich text

Auto-complete

Enable Auto-complete

Automatically suggest matches

Ignore case

Auto-indent

Enable auto-indent

Indent after newline

Tab auto-indents: Never

Trigger characters: #{}

Indent Braces Indent Blocks

Vertical Caret Policy

Even Jumps Strict Slop

Slop Value: 4

Unused Entities

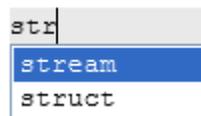
Highlight Unused Entities

Color:

- “Color on White” prints colored code on a white background regardless of the current display appearance.
- **Wrap Mode:** Choose the wrap mode you want to use for printing. The default is to wrap words to the next line, but you can choose to truncate lines or wrap at the character level, which breaks words across lines. The line breaks displayed are for printing only; no actual line breaks are added to your source file. See *Line Wrapping* on page 165 to change the wrap mode for screen display.

The **Include line numbers in rich text** option in the **Copy-and-paste** area lets you paste line numbers (in bold) when copying and pasting code into a word processor. The word processor must be able to handle Rich Text Format (RTF), which was developed by Microsoft. This option is off by default.

The **Auto-complete** options provide for auto-completion of keyword and entities you type in the editor. As you type, words are shown below your text. You can arrow down through the list and press Enter to choose a suggestion.



- **Enable Auto-complete:** This box is unchecked by default. If you want to enable auto-completion, check this box.
- **Automatically suggest matches:** If this box is checked, suggestions automatically appear below your typing. If you uncheck this box, you can still see and choose from a list of auto-completion options by pressing Esc while typing.
- **Ignore case:** If this box is checked, suggestions include upper and lowercase versions of the text you are typing.

The **Auto-indent** options allow you to control how tab characters are automatically added to code. If you check the **Enable auto-indent** box, automatic indentation happens as you type in the Source Editor. This smart indenting is currently implemented for C/C++, C#, Java, Javascript, and Perl code.

- **Indent after newline:** If this box is checked, when you start a new line, tabs are added so that you begin typing directly below the first character in the previous line. If you uncheck this box, the cursor is always in the first column when you start new lines.
- **Tab indents:** If this field is set to **Never** (the default), the <Tab> key always inserts tab or space characters. If it is set to **Always**, the <Tab> key always adjusts indentation to the “correct” level. If it is set to **Leading Whitespace**, the <Tab> key causes the appropriate amount to indenting in leading whitespace and inserts tabs or spaces everywhere else.
- **Trigger characters:** If you type one of the specified characters, the indentation level for the current line is modified to the correct level based on parsing of the code. For example, a “{” increases the indentation level, and a “}” decreases the indentation level. You can press Ctrl+Z to undo an automatic indentation that just occurred. The default trigger characters are # : { }

- **Indent bare braces:** If you set this value to greater than zero, the automatic indenting knows how to format code as in the following example, where the **Indent bare braces** value has been set to 2 and the **Indent width** is 4:

```
if (foo)
  {
    do_foo();
  }
```

The **Vertical Caret Policy** fields let you control how the Source Editor scrolls as the text cursor or current location highlight moves up and down. You can use these fields to optimize the amount of context you see when the Source Editor jumps to a new location. Most users will not need to modify these settings. If you are curious, you can see the descriptions of interactions between these fields at http://www.scintilla.org/ScintillaDoc.html#SCI_SETYCARETPOLICY.

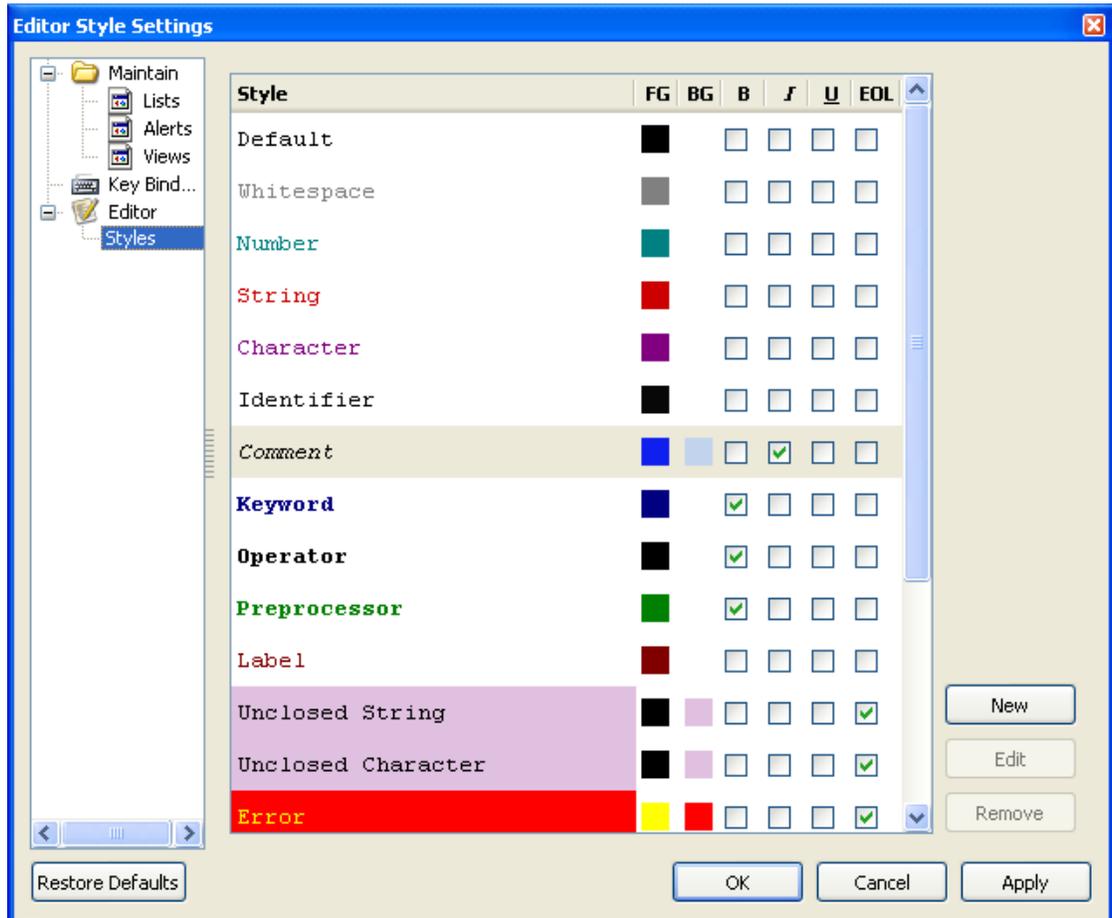
- **Even:** Checking this box causes the source code to scroll the same way both up and down.
- **Jumps:** Checking this box causes code to scroll multiple lines as needed to show some context for the current line of code.
- **Strict:** Checking this box specifies that you don't want the text cursor to go into the zone defined by the Slop Value. If Slop is unchecked, code scrolls to keep the current line in the middle of the window.
- **Slop:** Checking this box lets you define the number of lines at the top and bottom of the Source Editor which you do not want the text cursor to enter.
- **Slop Value:** This field lets you set a number of lines at the top and bottom of the Source Editor that the text cursor should avoid.

The **Unused Entities** fields let you choose whether to display entities that are never used with a colored background. By default, this feature is off. If you turn this feature on, the default background is gray for code that defines an unused entity. For example, if a function is never called, all the code in that function is highlighted when you enable this feature.

Editor > Styles Category

You can customize the colors used in the Source Code Editor in the Options dialog. To open this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Styles** category.

To change a color, click a color square next to an item in the list. Use the Select Color dialog to choose a new color for that item.



You can change the text foreground (FG) and background (BG) colors for any item. You can also make the text bold (B), italic (I), or underlined (U) for any item. To highlight the whole line for an item, check the EOL box.

By default, the following color codes are used for the source code:

- **Dark blue text:** Used for language keywords
- **Red text:** Used for characters and character strings
- **Italic blue text:** Used for comments
- **Green text:** Used for preprocessor statements
- **Black text:** Used for all other source text and for line numbers
- **White background:** Used for most source text
- **Pink background:** Used for inactive lines of code

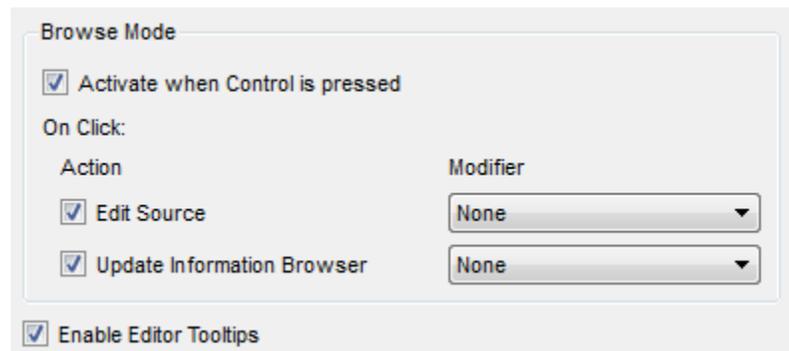
- **Gray background:** Used for line numbers
- **Yellow background:** Used to highlight text in Find Results for Find in Files

Additional items are available for customization depending on your source code language. For Delphi, you can customize the colors of module, routine, and type names. For FORTRAN, you can customize the colors of block, module, subprogram, and type names. For Ada, you can customize the colors of package names, subprogram names, and type names.

To create additional categories, click **New**. In the User Style dialog, type a name for the style, select the language to which this style applies, and type keywords to be highlighted in this style. Separate the keywords with spaces, line breaks, or tabs. Then click **Save**. You can then set the formatting for your new style.

Editor > Navigation Category

You can control the behavior of Browse Mode (see page 160) in the Source Editor. To see this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Navigation** category.



- **Activate when Control is pressed:** If this box is checked (on by default), Source Editor windows use Browse Mode if you are holding down the Ctrl key when pointing at an entity.
- **Edit Source:** If this box is checked (on by default), clicking an entity while in Browse Mode causes focus to jump to the declaration of that entity. You can choose a key (none, Alt, or Shift) that must be pressed along with the click to have this action occur. By default, you must press the Alt key when clicking to jump to the declaration of an entity.
- **Update Information Browser:** If this box is checked (on by default), clicking an entity while in Browse Mode causes the Information Browser to show information about an entity when you click on it. You can choose a key that must be pressed along with the click to have this action occur. The default is that no key is required along with the click.
- **Enable Editor Tooltips:** Check this box if you want to see brief information when the mouse cursor hovers over an entity name in source code. The information may include the full name, the type for a variable, and parameters and return values for a function. These tooltips are on by default.

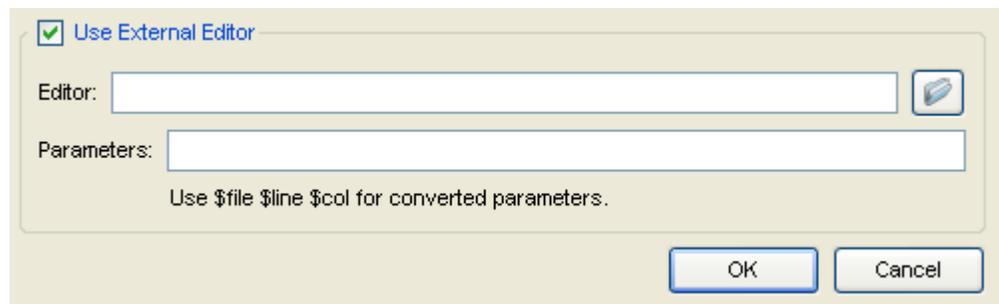
Editor > External Editor Category

You can use an editor other than the one provided with *Understand* for viewing and editing your source code. The editor you select is used whenever you open source code. This provides convenient source navigation while using a familiar editor. For example, you can use Microsoft Visual C++ or Emacs as your editor.

You should choose an editor that accepts command line parameters that specify the file to open, and a line and column number to go to.

To change the editor, follow these steps:

- 1 Choose choose **Tools > Options**. Expand the **Editor** category, and select the **External Editor** category.
- 2 In the Select an External Editor dialog, check the **Use External Editor** box if you do not want to use *Understand* for editing.



- 3 In the Editor field, click the folder icon and select the executable file for the editor you want to use.
- 4 In the **Parameters** field, type the command line parameters you want to use when opening the editor. Use the \$File, \$Line, and \$Col variables to allow *Understand* to open source files to the correct location.

For example, for the GVIM editor on UNIX, the **Editor** is “gvim”, and the **Parameters** should be as follows (for GVIM 6.0 or later):

```
--servername UND --remote +$line $file
```

For the TextPad editor on Windows, the **Editor** is most likely c:\Program Files\textpad4\textpad.exe, and the **Parameters** should be as follows:

```
$file($line,$col)
```

The *Understand* context menus (also called right-click menus) can be made usable in external editors. Steps for doing this are provided in the SciTools blog. For EMACS, vi, and Visual Studio, see <http://scitools.com/blog/2008/08/understand-context-menu-in-ema.html>. For SlickEdit, see <http://scitools.com/blog/2008/05/using-understand-with-an-exter-2.html>.

Graphs Category

The Graphs category of the **Tools > Options** dialog lets you control options related to how graphs are displayed. These options apply only to certain types of graphs, such as the Cluster Call and Cluster Call Butterfly graphs.



- **Highlight edges on hover:** Select this option if you want relationships within a graph to be highlighted when your mouse cursor hovers over a relationship. This makes it easier to distinguish between overlapping relationships.
- **On node/cluster double-click:** Controls what happens when you double-click on a node in a graph. By default, clusters are expanded or contracted. You can change this setting to show/hide relationships in one direction or the other. More options let you both expand/contract clusters and show/hide relationships at the same time.

Analyzing the Code

Once you configure a project, *Understand* can parse (that is, analyze) the project. During analysis, the source files are examined and data is stored in the *Understand* database. After parsing, the *Understand* database contains lots of data to browse.

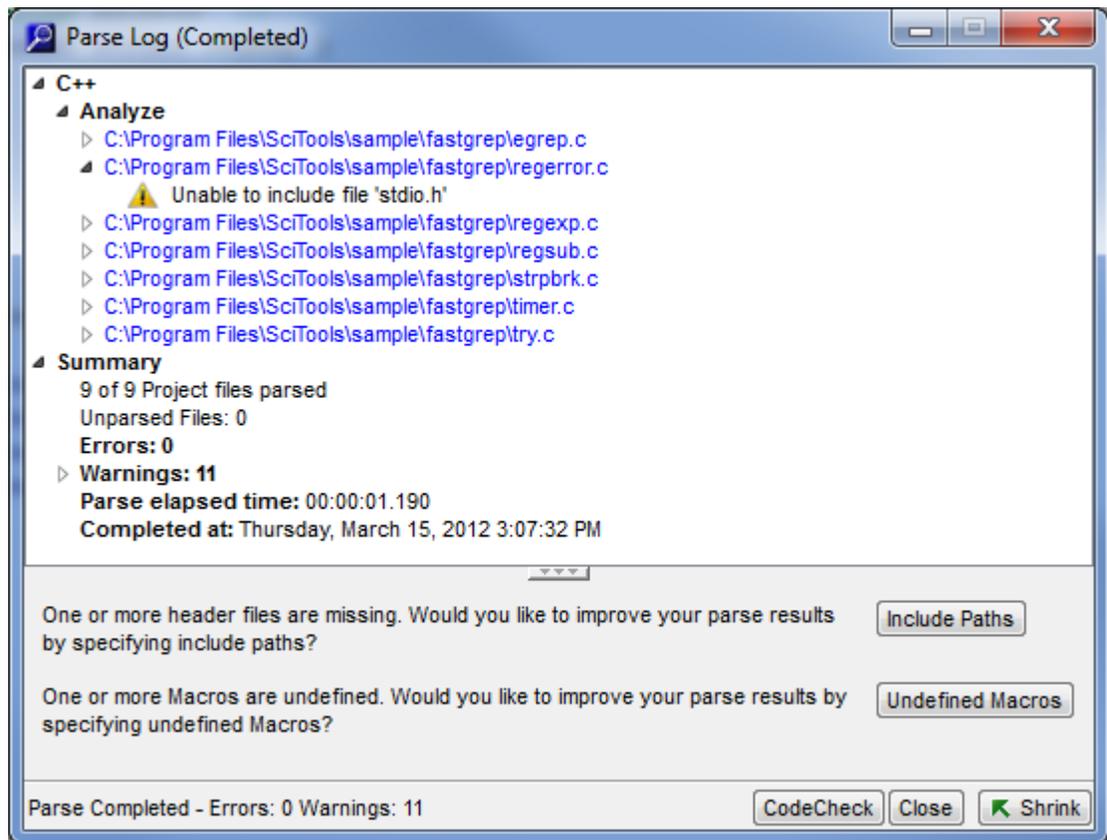
When you save or modify the project configuration, a prompt to analyze the project appears automatically. You can also analyze the project in the following ways:

Project > Analyze Changed Files: This menu command analyzes all files that have been changed and all files that depend on those changed since the last analysis. This is also referred to as “incremental analysis.” To analyze changed files, you can also left-click the toolbar icon shown here. (Ctrl+R)

Project > Analyze All Files: This menu command forces a full analysis of all project files, whether they have changed since the last analysis or not. (Ctrl+Alt+R)

As a shortcut for these commands, you can use the drop-down menu next to the toolbar icon.

For either command, the *Parse Log* window appears with a log of the results.



Analyzing a large project can take some time. If you click **Cancel** while the project is being analyzed you will see a message that says this action will leave the project in an incomplete state. You will need to analyze the project in order to explore it.

The parse log lists unparsed files, errors, and warnings.

If your project results in errors or warning that you did not expect, you should revisit the categories of the *Project Configuration Dialog* on page 39 to make sure your project is configured correctly. The cause of multiple similar errors is often something like not specifying an include file directory.

If some include files could not be found, click the **Include Paths** button. See page 116 for how to use the Missing Header Files tool.

If some macros were undefined, click the **Undefined Macros** button. See page 118 for how to use the Undefined Macros tool.

When the analysis is complete, the source code for any errors or warnings may be examined by double-clicking on the message in the *Parse Log* window.

To save the Parse Log to a text file, right-click on the white background of the Parse Log and choose **Save As**. Specify the location and name of the file you want to save. Or, you can use **Copy to Clipboard** to paste the parse log into another application.

If you have parsed the project during this session, you can choose **View > Last Parse Log** command to reopen the log.

The **CodeCheck** button opens the CodeCheck window. See Chapter 11, “Using CodeCheck for Standards Verification” for details.

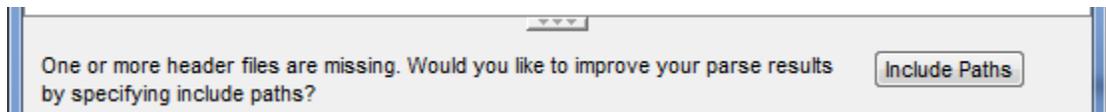
Tip: A configured project may be analyzed in batch mode using the command line program “und”. Refer to *Using the und Command Line* on page 298 for details on using “und”.

See *Analyze Category* on page 101 for options that affect the project analysis.

You can schedule automatic project analysis. See *Scheduled Activities* on page 50 for details.

Using the Missing Header Files Tool

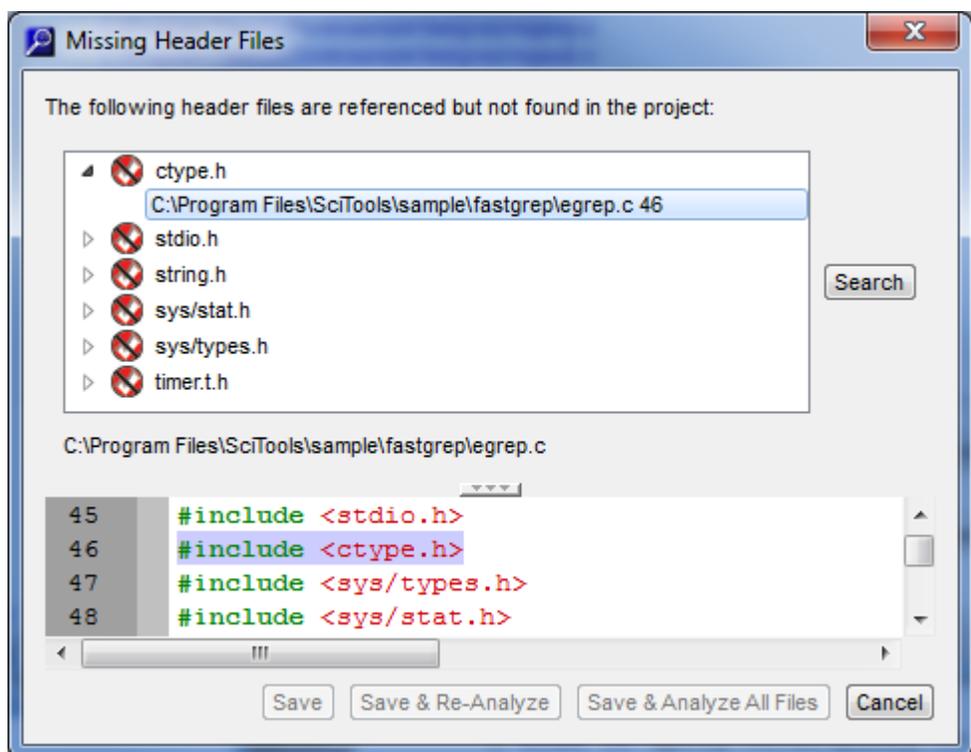
An **Include Paths** button is shown in the Parse Log if any header files are not found.



The configuration of your include file directories is important to improving the accuracy of project analysis.

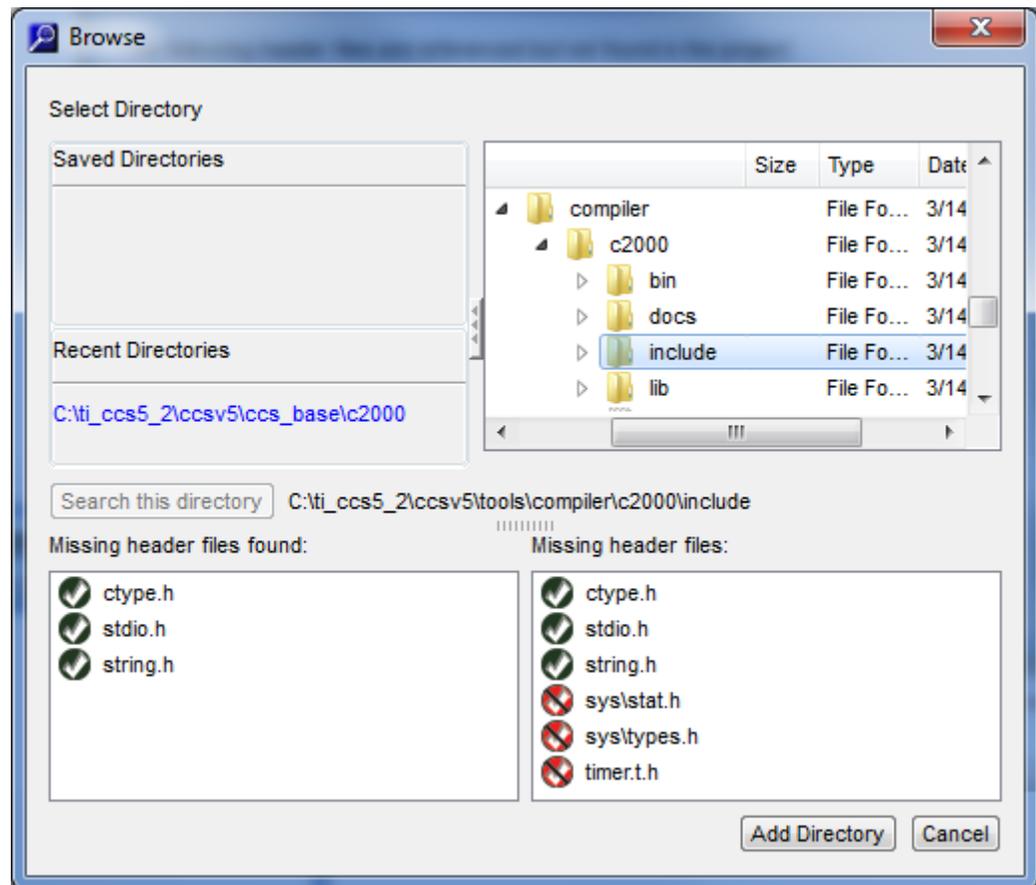
To use the Missing Header Files tool, follow these steps:

- 1 Click the **Include Paths** button to open the Missing Header Files dialog.



- 2 Expand the list for a missing header file and select the path to a source file. You will see the code that includes this missing file.

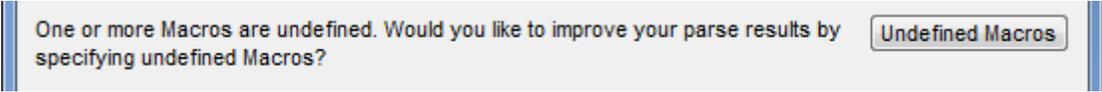
- 3 Click the **Search** button and browse to find the location of the file.



- 4 If you do not know the specific directory, you can browse a directory that you think contains the include files and click **Search this directory**.
- 5 Once you have found the directory, click **Add Directory**.
- 6 In the Missing Header Files dialog, either click **Search** to find more directories or click one of the **Save** or **Save & Analyze** buttons.

Using the Undefined Macros Tool

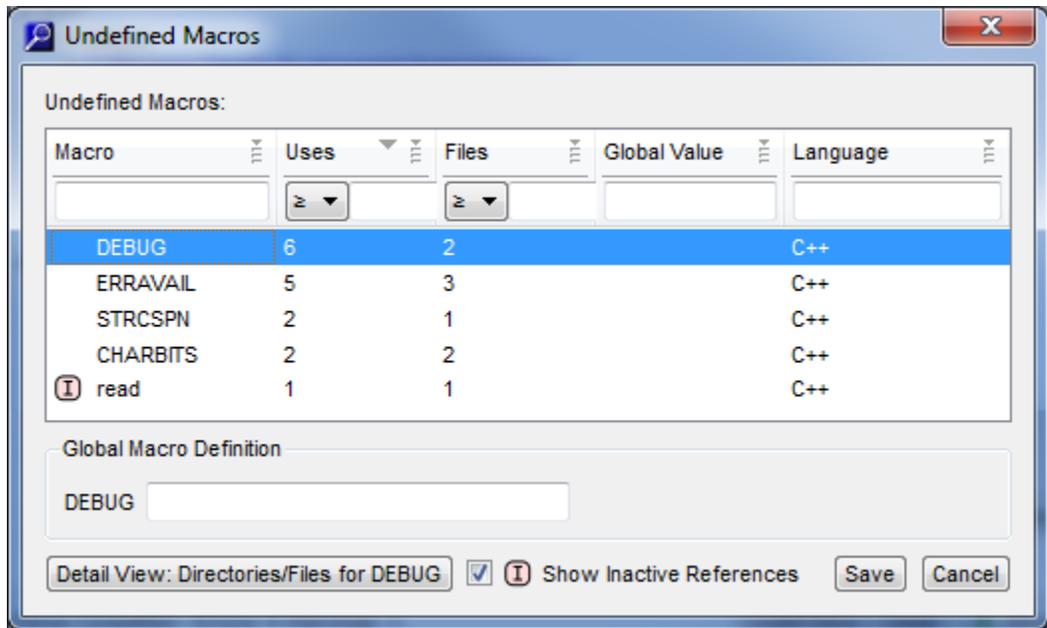
An **Undefined Macros** button is shown in the Parse Log if any macros are not defined in the code or project configuration.



The configuration of your include file directories is important to improving the accuracy of project analysis.

To use the Undefined Macros tool, follow these steps:

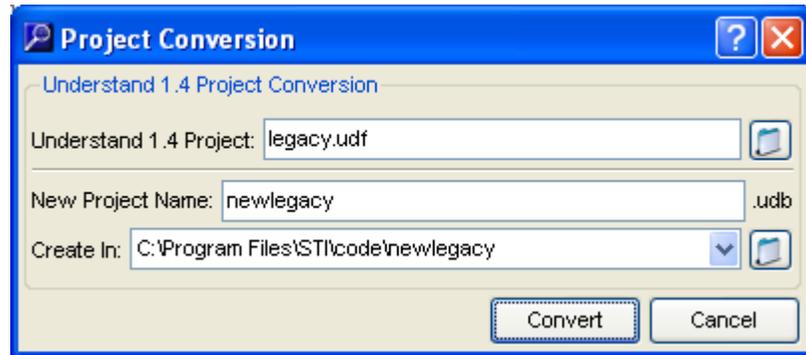
- 1 Click the **Undefined Macros** button to open this dialog:



- 2 Select a macro. You can use the headings and fields at the top to sort and filter the list and the **Show Inactive References** box to show or hide such macros. See *Filtering the List* on page 149 for more about using these filter fields.
- 3 Type a value for the macro in the Global Macro Definition area, and click **Save**.
- 4 You can click the **Detail View** button to see the code where the selected macro is used. In this view, you can define a macro value for a specific file or folder instead of project-wide.

Converting an Understand 1.4 Project

You can open and convert an *Understand 1.4* project to an *Understand* project. To do so, choose **File > Open > Understand 1.4 Project** from the menus.



In the **Understand 1.4 Project** field, click the folder icon to browse for the database file for your *Understand 1.4* project. This file will have a file extension of .uda, .udc, .udf, .udj, .udv, or .udp, depending on the programming language used in the project.

In the **New Project Name** field, type a name for your new project. The file extension of .udb will be added automatically. This may be the same name as the old project, but need not be.

In the **Create In** field, click the folder icon to browse for the directory in which you want the new project stored. The default is the directory that contains the *Understand 1.4* project.

When you click **Convert**, you see the Parse Log window for the project being converted.

This chapter covers the basic windows in *Understand* and their options in detail. It also covers operations within the Filter Area and the Information Browser.

Details on the use and operation of the Entity Locator and Find in Files for searching for and locating entities are provided in the chapter *Searching Your Source* on page 139.

Details on the use and operation of the **Source Editor** is contained in the chapter *Editing Your Source* on page 157.

This chapter contains the following sections:

Section	Page
PLEASE RIGHT CLICK	121
Various Windows Explained...	122
Entity Filter	123
Information Browser	125
Project Browser	130
Exploring a Hierarchy	132
Dependency Browser	133
Favorites	135

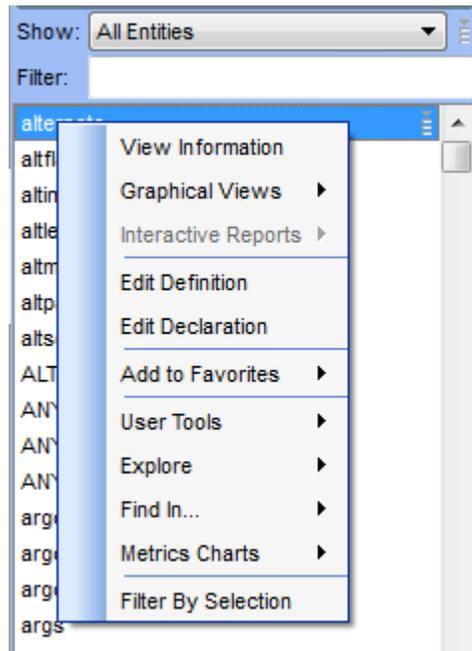
PLEASE RIGHT CLICK

Sorry for shouting (by using all caps above). In order to make the *Understand* interface as quick, tight and elegant as possible, we have hidden a lot of power beneath your mouse buttons.

The general rule is that anywhere you look you can right-click to do or learn something.

A second general rule is that right-click reuses windows where it can and **Ctrl + right-click brings up new windows.**

So please right-click. There will be no more reminders.



Check out all the stuff you can learn or do right-clicking!

Right-click almost anywhere to bring up a menu.

Ctrl + right-click brings up the same menu but actions happen in a new window.

Various Windows Explained...

Understand's GUI has a number of tools for locating and examining entities. This chapter provides a brief list of all these tools and describes the Entity Filter, Information Browser, and Favorites in detail.

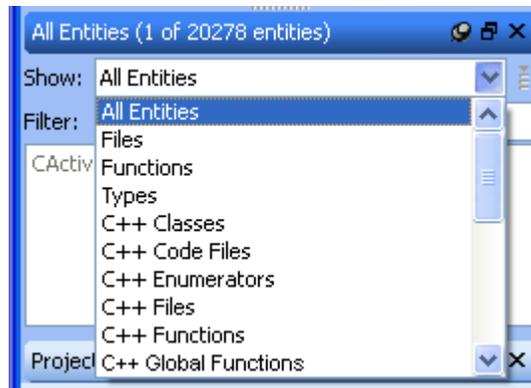
The tools available for finding and exploring entities are:

- **Entity Filter:** Provides an alphabetic list of entities of the selected type. See page 123.
- **Information Browser:** Provides an explorer for entity characteristics and connections. See page 125.
- **Project Browser:** Lets you browse a hierarchical file list. See page 130.
- **Exploring View:** Lets you browse a relationship hierarchy. See page 132.
- **Dependency Browser:** Lets you browse dependency relationships. See page 133.
- **Favorites:** Lets you provide quick links to frequently-used entities. See page 135.
- **Entity Locator:** Lets you filter all entities in a project in complex ways. See page 148.
- **Find in Files:** Searches multiple files. See page 143.
- **Source Editor:** Shows source code. See page 157.
- **Contextual Information Sidebar:** Show context information about the current source editor file. See page 156.
- **Scope list:** Lists the functions or similar constructs in a file. See page 159.
- **Architectures:** Defines named regions and views of the project. See Chapter 7.
- **Graphical Views:** Shows connections and structures of entities. See Chapter 10.
- **Reports:** Generate reports about entities. See Chapter 8.
- **Metrics:** Generate statistics about entities. See page 209.

Entity Filter

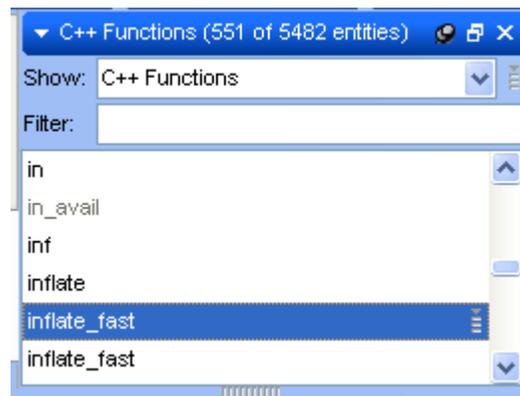
The *Entity Filter* provides a quick list of the selected entity type. You can filter this list to match a text string.

The options in the **Show** list depend upon the languages you have enabled for your project and the types of entities and relationships found in your project. If your project uses multiple languages, the language is listed along with the type.



Note: For especially large projects, the All Entities option may be disabled to prevent memory errors.

For each of the entity types, you can quickly find any entity that has been declared (or used) in the source code.



By default, the entities are sorted in ascending (A to Z) order. You can reverse the order by clicking the drop-down icon  and choosing **Sort Descending**.

You can only have one Entity Filter open. If you close the Entity Filters window, reopen it by choosing **View > Entity Filter**.

Using the Filter Field

In the **Filter** field, you can type a string to match a set of entities. Entity names match if the string is contained anywhere in the name. So, for example, you can type “y” to list only entities that contain a Y or y anywhere in the name.

By default, filtering is case-insensitive. You can make it case sensitive by clicking the drop-down icon  and choosing **Filter Case Sensitivity > Case Sensitive**.

If you want to quickly jump to the point in the list where entities begin with a particular letter, just click in the list of entities and type a letter.

You can select other ways for the **Filter** field to work. Click the drop-down icon  and choose **Filter Pattern Syntax**. The options are:

- **Fixed String:** This is the default behavior.
- **Wildcard:** With this option selected, you can use * (any characters) and ? (any single character) wildcards for pattern matching. See page 149 for examples.
- **Regular Expression:** With this option selected, you can use UNIX-style regular expressions. See page 149 for an overview.

When you are finished using a filter and want to see all the entities for the selected type, click the drop-down icon and choose **Clear Filter**.

If you change the type of entity in the **Show** field, any filter you have typed is cleared if the **Clear Filter Text on Filter Type Changes** option is selected in the menu available from the drop-down icon .

Customizing the Display

You can modify how the Entity Filter lists entities as follows:

By default, the full entity name is shown in the Entity Filters list and entities are alphabetized by their full name. This name may include a class prefix or other language-specific prefix type. To list entities by their “short”, unprefixed names, click the drop-down icon and choose **Entity Name as > Short Name**.

By default, only the name of the file is shown in a Files list in the Entity Filter. This name does not include the file location. To list files including their locations, click the drop-down icon and choose **File Name as > Relative Name** or **File Name as > Long Name**.

Root Filters

Notice that there are the filter type names that contain “Root”, as in **Root Calls**, **Root Callbys**, and **Root IncludeBys**. These “Root” types show only the top of a given tree. The tops (or bottoms) of relationship trees are often helpful points to begin exploring code that is new to you.

- **Root Calls:** Lists only entities that call others, but are not called themselves. These are either high-level code (mains), code called by hardware (interrupt handlers), or dead (unused) code.
- **Root CallBys:** Lists only entities that are called by others, but that do not call anybody else. These are low-level routines.
- **Root IncludeBys:** Lists only files included by others, but not included themselves. These are “lower” level include files.

- **Root Classes:** Lists only classes not derived from other classes. These are candidates for lower level classes, or library classes.
- **Root Decl:** Lists only the highest level declaring routines. (Ada)
- **Root Withs:** Lists only program units (packages, tasks, subprograms) that With other program units, but are not withed by anybody else. (Ada)

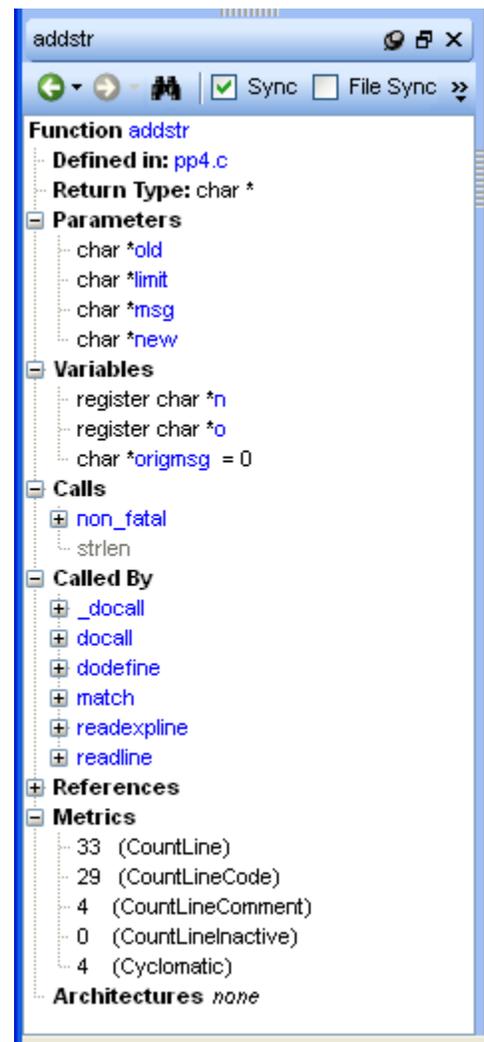
Information Browser

When you click on an item in the *Entity Filter* or in a number of other windows, the *Information Browser* updates to show everything that *Understand* knows about that entity. The *Information Browser* shows this data as a tree whose branches can be expanded individually or all at once.

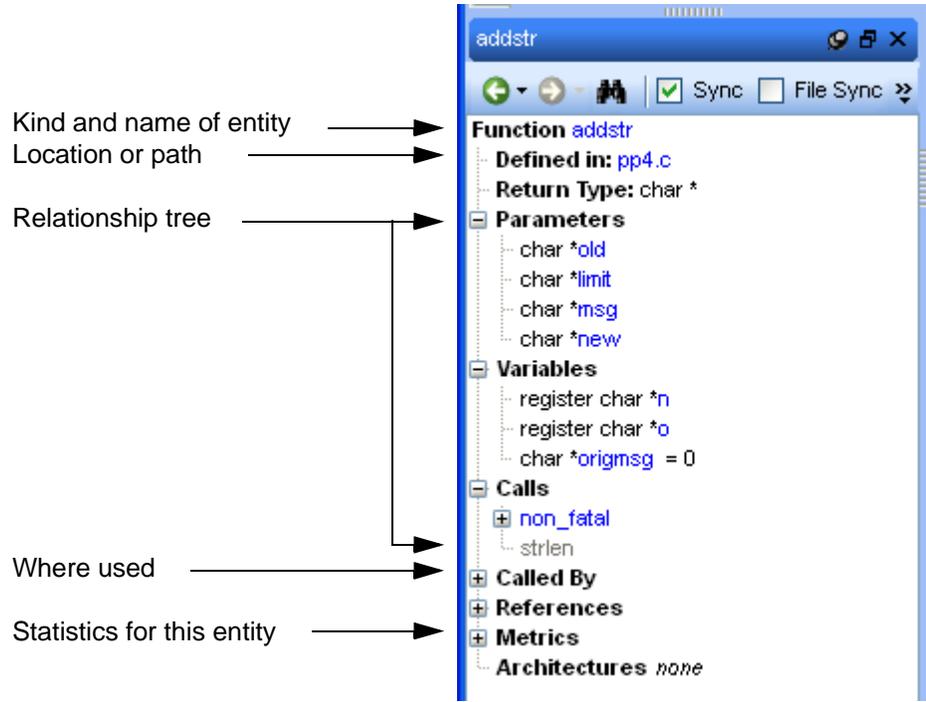
If the *Information Browser* isn't open, you can open it by either clicking on an item in the *Entity Filter* or *Project Browser*. You can also right-click on an item anywhere and choose **View Information**. Or, choose **View > Information Browser** from the menus.

Everything *Understand* knows about an entity can be learned using the *Information Browser*. The information is shown in a tree. The tree can be expanded selectively or in bulk. Each terminating item (leaf) of a tree provides some information about that entity.

All information in an *Information Browser* window can be saved to a text file, or copied and pasted via standard Windows or X11 copying functions.



As you drill down you can change which entity you are learning about. Each time you change the entity, it is remembered in the Information Browser history for quick backtracking.



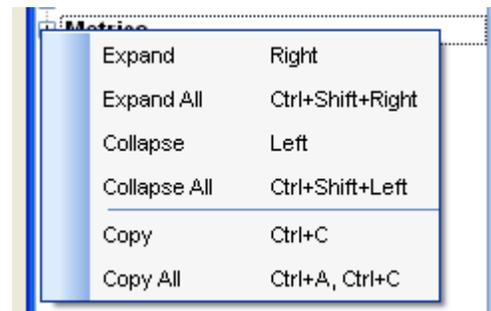
Drilling Down A Relationship

Drilling down the tree works as expected (mostly). To expand a tree, click on the + sign. To close the tree click on the - sign.

Right-clicking brings up a menu that includes expand/collapse options. **Expand All** provides a shortcut to expand all levels of the selected branch.

To open or close the entire tree, right-click on the top item and choose **Expand All** or **Collapse All**.

See *Saving and Printing Information Browser Text* on page 129 for details on the other options in this context menu.



Displaying More or Less Information

If you click the  icon next to a bold heading such as **Calls**, **Called By** or **References** in the Information Browser (or right-click on the heading), you'll see options that let you modify how that entity is listed. These options include:

- **Fullname:** If checked, the fully-qualified name of the entity is shown.
- **Parameter:** Lists the parameters.
- **Reference:** Choose "Full" to include the file and line location of the reference.

- **Return Type:** Lists the return type.
- **Sort:** Controls the sort order of the list.
- **Type:** If checked, the datatype is shown.
- **Filename:** Controls whether the reference format is short, long, or relative to the project database.
- **Group by:** For C++ classes, you can choose whether to sort class members by the type of access available (public or private) or the kind of member (function or object).

Searching the Information Browser

If you click the binocular icon at the top of the Information Browser (or click in the Information Browser and press Ctrl+F), a Find bar appears at the bottom of the Information Browser.

Type text in the box and click a forward or backward arrow to find an occurrence of the string in the Information Browser text. All text is searched, including node names and items that are currently hidden by collapsed nodes. If you type a string that does not appear anywhere in the Information Browser text, the field turns red.



To make the search **Case Sensitive** or to match only **Whole Words**, use the drop-down arrow to select those commands.

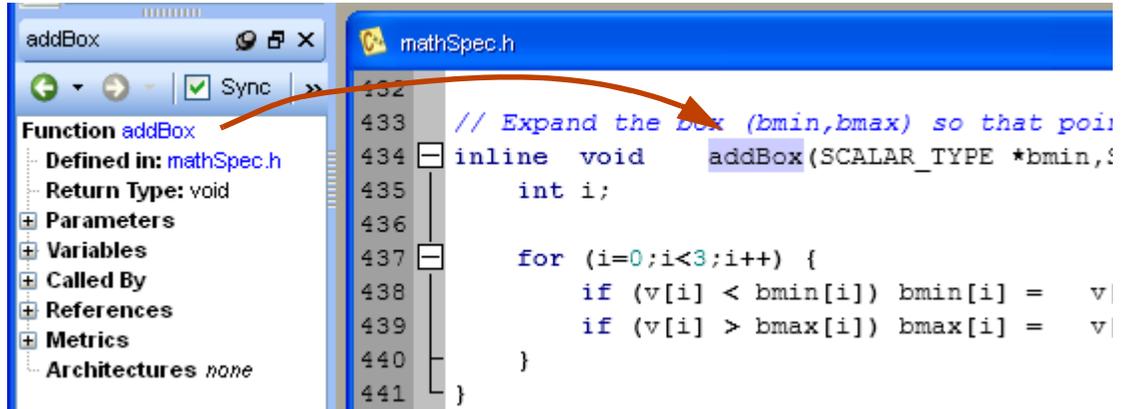
Syncing the Information Browser

You can have multiple Information Browser windows open if you uncheck the **Sync** box. Selecting an entity or choosing **View Information** updates the Information Browser that has its **Sync** box checked.

The **File Sync** box synchronizes the Information Browser with the file in the active Source Editor.

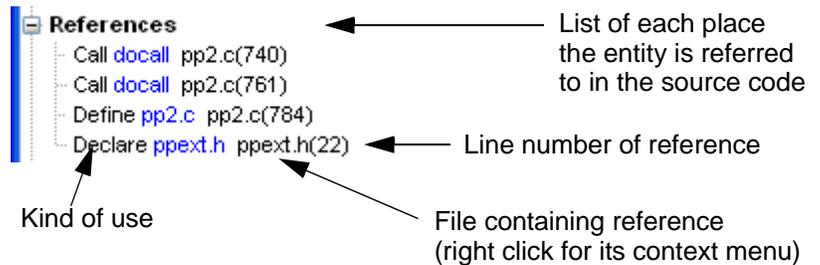


Visiting Source Code In general, if you double-click on an entity in an informational window (*Information Browser* or *Entity Filter*) the declaration of that entity will be loaded into the *Document Area*.



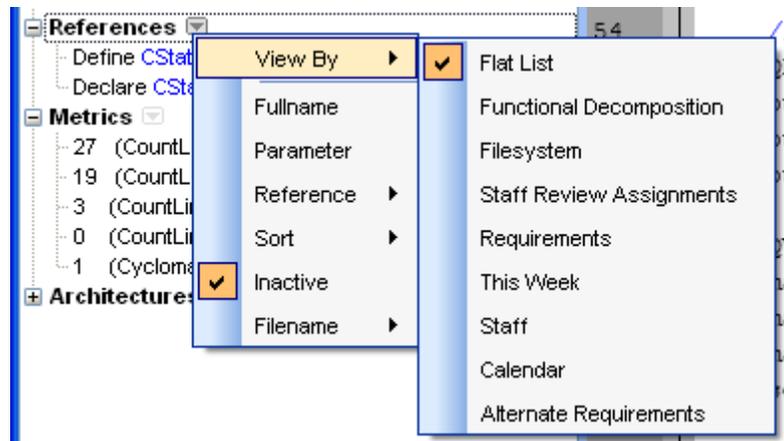
Another way to visit source from any entity you see in *Understand*, is the context menu. Where appropriate, an entity's context menu contains an **Edit Source** (Ctrl+Shift+S) menu item. In some cases, there are separate menu items for **Edit Definition** and **Edit Declaration** (Ctrl+Shift+D) or separate menus for other language-specific locations. If you have a .c or .h file selected, the **Edit Companion File** command opens the other file if one exists for that filename.

Visiting References The portion of the *Information Browser* labeled "References" lists everywhere the entity is referred to in the analyzed source code:



Left-click on any reference to visit that location in the source code.

Right-click on the “References” title for the node or the down-arrow next to the node to choose how to organize the references. Your choices are the default flat list, and all of your architectures.

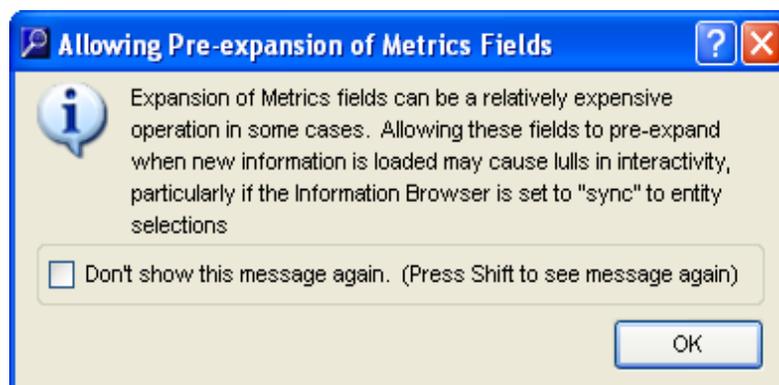


Viewing Metrics

The last node on the Information Browser tree is **Metrics**. This branch shows the metrics available for the current entity.

By default, when you switch to another entity in the Information Browser, the Metrics node is closed automatically. This is because it can take a long time to update the metrics for each entity in a large project.

If your project is small enough that updating metrics as you switch between entities does not take a long time, you can right-click on the **Metrics** node and choose **Allow Pre-expansion**. The Metrics node will then stay open when you change entities. You see the following warning about the time required for metric updates.



See *Metrics Reports* on page 209 for details on metrics.

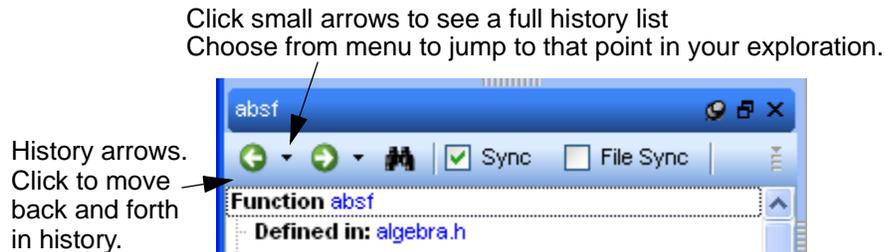
Saving and Printing Information Browser Text

All text shown in the *Information Browser* can be copied to the clipboard for pasting into another application as unformatted text. Only the currently expanded branches are pasted. When saving or pasting in text format, the branches of the tree are represented by indents in the text.

The context menu offers choices to **Copy** (only the selected line) and **Copy All**.

Entity History

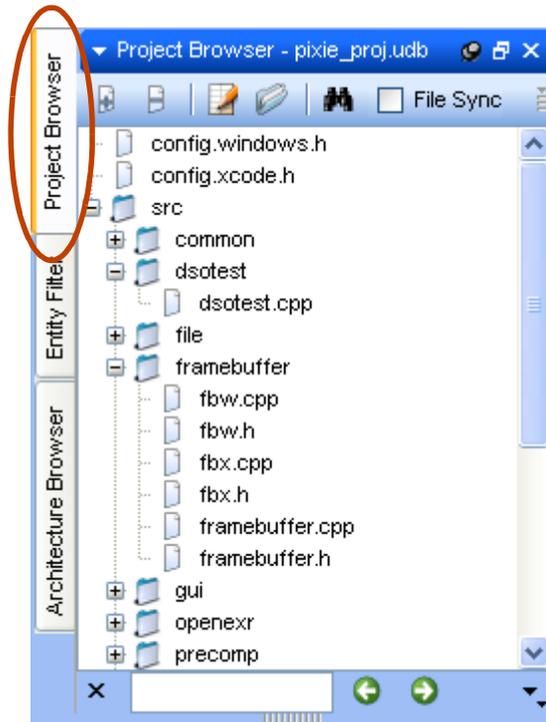
As you explore your code, you can go many places quickly. Often you want to backtrack to explore a new path. To help you do this, the *Information Browser* contains a full history of what it has displayed. The *Information Browser* history can be found in the upper-left corner:



Use the right and left arrows to move back and forward in the history list. The down-arrows show the whole list.

Project Browser

To open the Project Browser, choose **View > Project Browser** from the menus.



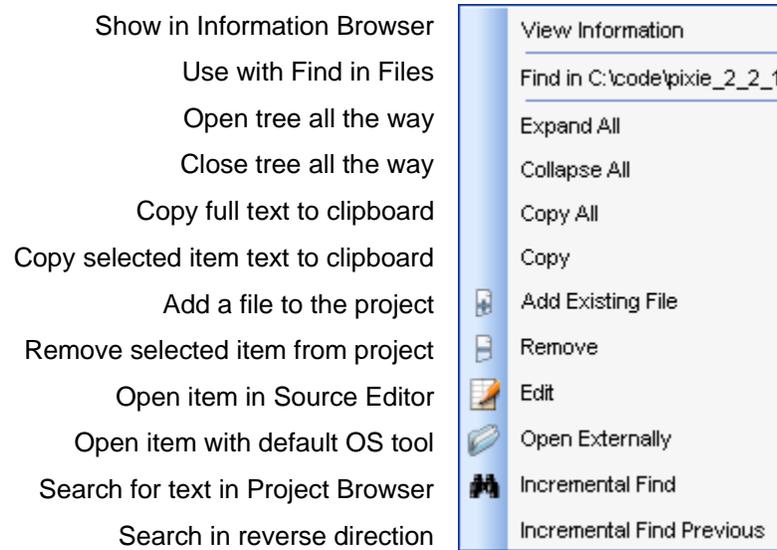
By default, the Project Browser is in the same area as the Entity Filter (and the Architecture Browser). Use the tabs on the left to switch between browser tools in this area.

The Project Browser shows the project files in their directory hierarchy. You can expand and collapse the tree as needed.

Press Ctrl+F to display a search line at the bottom of the Project Browser.

The **File Sync** box synchronizes the Project Browser with the file in the active Source Editor.

The context menus for this view offer a number of options. The options with icons are also available in the toolbar for the Project Browser.



For a file, the context menu includes additional commands, including commands to open graphical views, rename the file, parse (analyze) this file only, find uses of the filename in project files, and add the file to the Favorites list.

The **Add Existing File** command lets you browse for and add source code files to the project.

To use the **Remove** command, select one or more files and folders and choose this command. The Confirm Project Modification dialog lists files that will be deleted from the project if you click **Yes**.

The **Open Externally** command opens an operating system dependent tool for the directory or file. For example, on Windows it opens a directory using the Windows Explorer. For a file it opens the default tool for the file extension.

The **Incremental Find** command opens a Find bar at the bottom of the Information Browser. Type text in the box and click an arrow to find an occurrence of the string in the Project Browser text. All text is searched, including files in folders that are currently closed. If you type a string that does not appear anywhere in the Project Browser, the field turns red. See page 127 for details on search options.

Exploring a Hierarchy

The Exploring view lets you browse up and down a relationship hierarchy within your project.



The context menu in the Information Browser, Entity Filter, and Project Browser offers commands to Explore certain types of entities. The command will be similar to **Explore > Explore Called By/Calls** or **Explore > Explore Includes**.

If you click on an item in one column, you see its relationships in the columns on either side. As you choose items to the left or right, columns resize to show more of the hierarchy. Calls and Includes go from left to right. Callbys and Includebys go from right to left.

If you double-click an item, a Source Editor window shows the entity's definition.

The **References** area shows the line number of the currently highlighted relationship. Double-click to visit that code.

If you check the **Generate Syncs** box, then the Information Browser automatically displays information about any entity you select in the Exploring window. Holding the Shift key down temporarily activates this behavior.

If you check the **Jump to First Reference** box, then the Source Editor automatically displays the initial reference to any entity you select in the Exploring window. Holding the Ctrl key down temporarily activates this behavior.

Dependency Browser

The Dependency Browser lets you examine which items are dependent on others. You can use the Dependency Browser with architecture nodes, files, classes, packages, and interfaces.



To open the Dependency Browser, right-click on an architecture node, filename, class name, or package name anywhere in *Understand*, for example in the Entity Filter, Information Browser, or a graphical view. Choose **View Dependencies** from the context menu.

The left panel shows the item you selected and the items it contains. The right panel shows items that either depend on the item selected in the left panel or are dependent on that item, depending on your selection in the **Dependency Kind** field. For example, an item depends on another if it includes, calls, sets, uses, casts, or otherwise refers to that item.

You can expand hierarchies in the left and right panels. For example, when you view dependencies for an architecture node, you can expand it to see lower-level architecture nodes, then files, then the entities in the files. Letters next to items identify whether they are architecture nodes ("a"), files ("f"), classes ("c") or entities in files ("e").



You can use the **Group Results By** field to select an architecture to control how the items in the right panel are organized.

If the **Files** box is checked, the results in the right panel are also organized according to the files that contain the dependencies. Files are listed below the lowest-level architecture node that applies.

If the **Classes** box is checked, the results in the right panel are also organized according to any classes that contain the dependencies. Classes are listed below the file level if both boxes are checked.

If the **Entities** box is checked, the results in the right panel are also organized according to the entities that contain the dependencies (for example, functions or objects). Entities are listed below the class level if both boxes are checked.

Click the  icon to create a graphical view of the dependencies currently shown in the Dependency Browser. While the Dependency Browser shows one level of dependency, graphical views can show multiple levels. See *Viewing Architecture Dependency Graphs* on page 179 for details.

Click the  icon to export a comma-separated values (CSV) report of dependencies for the top item in the left panel of the Dependency Browser.

You can also use the mouse to select items in the right panel of the Dependency Browser. Then right-click on a letter icon and choose **Copy** to place the currently selected text on your clipboard for pasting into other applications.

If the **Reuse** box is unchecked, a new Dependency Browser is opened when View Dependencies is selected for another entity or node. The Reuse box is checked by default.

If the **Sync** box is checked, the Dependency Browser automatically displays information about any architecture node, file, class, or package you select in the Project Browser, Entity Filter, Architecture Browser, or similar window. This box is unchecked by default.

Click the drop-down icon  if you want to change any of the following display options:

- **Architecture Name.** The default is to show the relative name, but you can select short name or long name instead.
- **Entity Name.** The default is the short name, but you can select the long name instead.
- **File Name.** The default is the short name, but you can select the relative name or long name instead.

See *Dependency Category* on page 104 for information about controlling what types of relationships are treated as dependencies.

See *Exporting Dependency Metrics* on page 225 for other ways to export information about dependencies.

Favorites

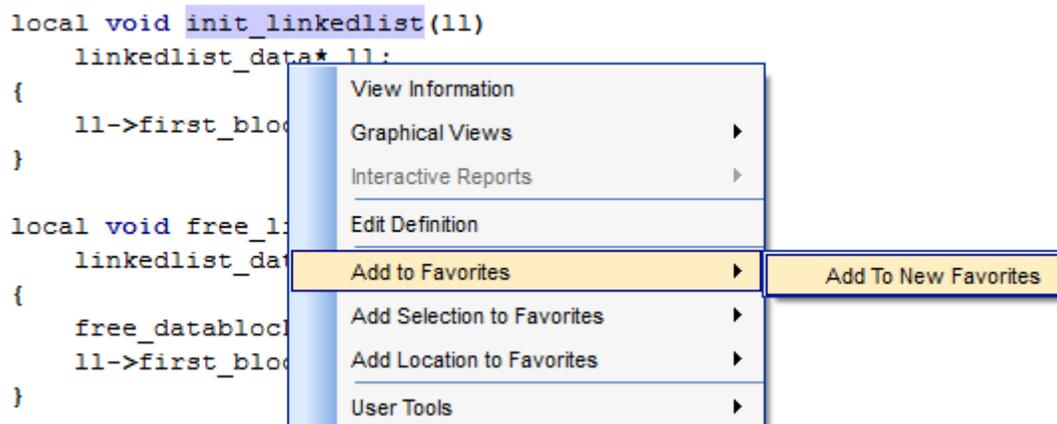
You can mark all kinds of things in Understand as “Favorites” so that you can quickly access them as you would web pages in a browser’s Favorites group. Your favorites can include entities, code locations, graphs, Information Browser displays, and dependencies. You can also store multiple plain text strings in your favorites, so that you can quickly copy one of your saved strings to the clipboard.

Favorites are saved as part of a project.

Creating a Favorite Entity

To mark an entity as a favorite, follow these steps:

- 1 Select an entity name and right-click in source code, the Entity Filters area, the Information Browser, a graphical view, or anywhere else entities occur.



- 2 Choose **Add to Favorites > Add To New Favorites** (or an existing favorites group). This adds a link to the entity itself, even if the line number changes later. If you’ve already created a favorites group, you can select it from the submenu instead of using **Add To New Favorites**.
- 3 Alternately, if you right-click on a source file, you can choose **Add Location to Favorites** to add the line number in the file to a favorites group.

See *Creating a Plain Text Favorite* on page 138 for information about the **Add Selection to Favorites** command, which stores text for pasting from the clipboard.

- If you choose to create a new favorites group, you see the New Favorite dialog. Type a name for the new group and click **OK**.



- When you create a favorite, the Favorites group opens.

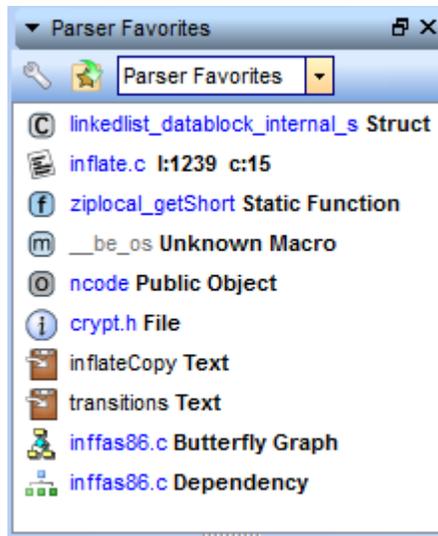
Creating a Favorite View

Besides entities and code locations, favorites can include graphical views, Information Browser views, and Dependency Browser views.

To add a favorite for any of these items, click the **Favorites** icon in the toolbar for the view. By default, the view is added to the last favorites group you used. If you want to place it in a different group, choose a group from the drop-down menu.

Using a Favorites Group

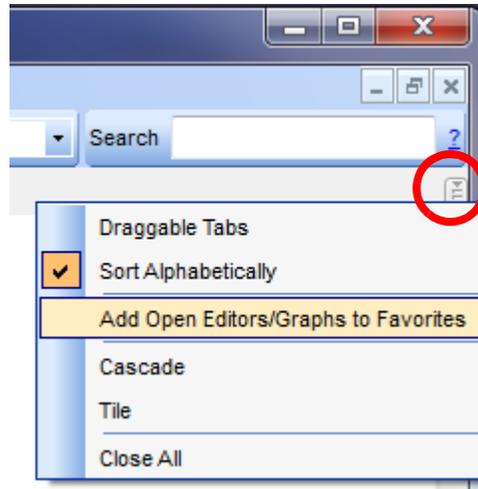
To open a Favorites group, choose **View > Favorites** and choose a group name from the menus. You see the favorites saved to that group.



In the Favorites view, you can use the drop-down list to switch to a different Favorites list.

Click on a link in the favorites group to jump to that location. You can open all the favorites in the current list by clicking the **Open Favorites** icon.

You can add all the currently open files and graphical views to a Favorites list by clicking the small dimple icon in the upper-right corner of the document area and selecting the **Add Open Editors/Graphs to Favorites** command.



As with just about every place in Understand, you can right-click on a favorite to see a context menu that includes commands such as View Information, Graphical Views, and Find In.

An icon to the left of each favorite (and the text after the name of the favorite) identifies each favorite's type. For example, in the previous figure the first five favorites link to various types of entities or line numbers in the code.

Favorites with a  icon link to a file.

Favorites with a  icon link to an Information Browser view.

Favorites with a  icon link to a Dependency Browser view.

Favorites with a  icon link to a graphical view.

Favorites with a  icon store text that you can paste into source code. See *Creating a Plain Text Favorite* on page 138 to store text as a favorite. When you click on a text favorite, the text is placed in your clipboard and you can paste it into Source Editor windows or other applications.

If you click the  **Configure** wrench icon at the top of a Favorites group, additional toolbar icons are displayed. These let you manage favorites you have already created as follows:

  The arrow icons move the selected favorite up or down in the group.

 Click this icon to create a header that you can use to organize your Favorites.

 Click this icon to create a text favorite. See *Creating a Plain Text Favorite* on page 138 for details.

 Delete the selected favorite from the group.

 Rename the current favorites group.

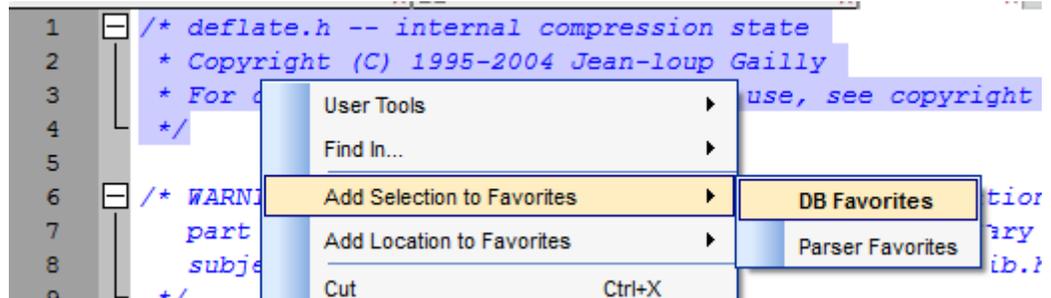
 Delete the current group of favorites.

Creating a Plain Text Favorite

You can store multiple plain text strings in your favorites, so that you can quickly copy a saved string to your clipboard and paste it as needed into your code. For example, you might use a standard comment at the beginning of files or elsewhere in your code.

To save text as a favorite, follow these steps:

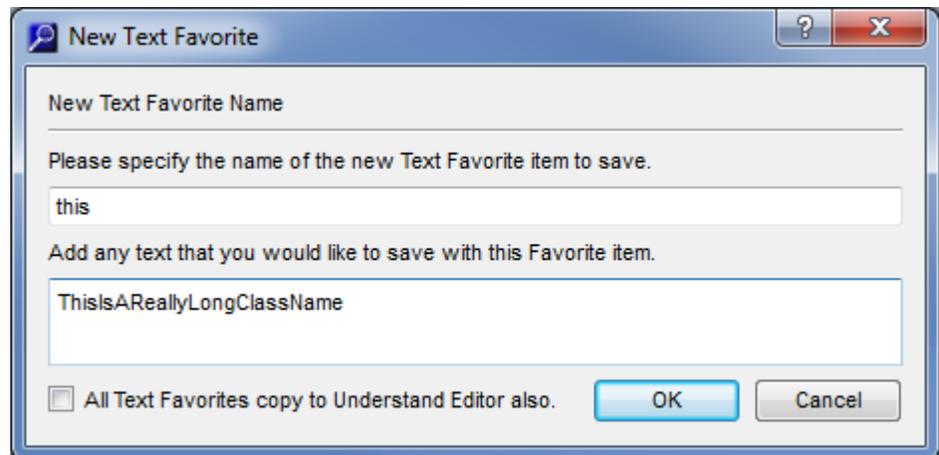
- 1 Select text in a Source Editor and right-click.



- 2 Choose **Add Selection to Favorites** and select a favorites group from the submenu.

- 3 When you create a favorite, the Favorites group opens.

You can also create a text favorite by clicking the  icon in the Favorites area (which you can see if you select the  **Configure** icon at the top of the Favorites area). You see the New Text Favorite dialog.



In the first field, type a short name to be shown in the Favorites list.

In the second field, type the full text of the favorite.

To use a text favorite, double-click on the name of the favorite in your Favorites list. This copies the longer text from the second field to your clipboard, so that you can paste it into a Source Editor.

If you check the **All Text Favorites copy to Understand Editor also** box, then when you click on a Text Favorite, the text you typed in the second field is automatically pasted into your current Source Editor window at the text cursor position.

This chapter covers how to use *Understand's* Find in Files and Entity Locator features to locate things in your source code.

This chapter contains the following sections:

Section	Page
Searching: An Overview	140
Instant Search	141
Find in Files	143
Entity Locator	148
Finding Windows	152
Searching in a File	155

Searching: An Overview

Finding things in large bodies of source code can be difficult, tedious, and error prone.

Understand offers a number of ways to search for strings in your source code or to locate particular lines. The commands for these options are located in the **Search** menu. These commands are described in the locations listed in the following table:

Search Command	See	Comments
Entity Locator	page 148	project-wide, entity-based
Find in Files	page 143	project-wide, text-based
Replace in Files	page 146	project-wide text-based
Instant Search	page 141	project-wide text-based
Find Next and Previous	page 155	single file
Find & Replace	page 155	single file
History	page 153	project-wide
Bracket Matching	page 163	single file
Favorites	page 135	project-wide
Contextual Information Sidebar	page 156	single file
Bookmarks	page 166	project-wide

Each of these searching methods has advantages and disadvantages. Together they provide powerful ways to easily find what you need to find to better understand and modify your code.

See page 122 for a more complete list of the code exploration tools in *Understand*.

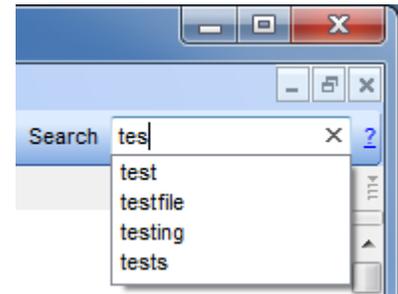
Instant Search

Instant Search lets you search your entire project instantly, even if it contains millions of lines of source code. As you type, you can see terms that match the string you have typed so far.

The search box for Instant Search is in the upper-right corner of the *Understand* window.

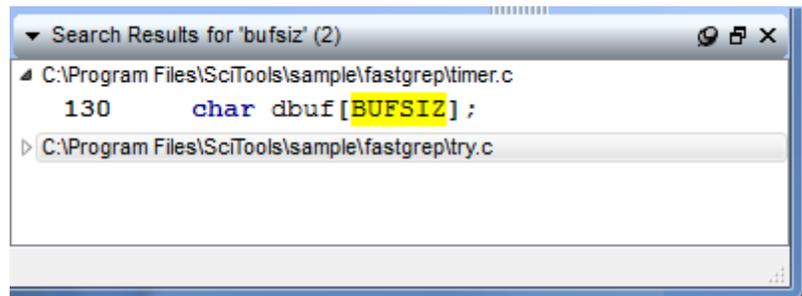
If you don't see this field, choose **View > Toolbars > Search** from the menus.

To begin searching, click in the Search field and type a string you want to find. You can also press Ctrl+Alt+S or choose **Search > Instant Search** from the menus to move your cursor to the Search field.



The easiest way to use Instant Search is to type a string that you want to match in your code. Press Enter after typing a search string to see a list of files that match your search in the Search Results area.

Right-click in this area to **Expand** or **Collapse** the results tree. Choose **Find** to use the **Previous** and **Next** icons to move through the results one-by-one. You can double-click on a file to open the file and see the line of code in the Search Results area.



A number of more powerful search options are supported with Instant Search. The syntax used by this field is based on the syntax used by Apache Lucene, an open-source text search engine library. See

http://lucene.apache.org/java/2_3_2/queryparsersyntax.html for syntax details.

The following list explains some of the syntax options available:

- Searching is case insensitive. A search for test also matches “Test” and “TEST”.
- Unless you use wildcards, searching matches whole words. A search for test does not match occurrences of “testfile”.
- The wildcards available are * (any number of letters and digits), ? (any single letter or digit). You cannot use a wildcard as the first character in a search string.

- When indexing the code (which happens in the background), Instant Search breaks code into searchable strings by splitting the code at white space and punctuation (and syntax conventions for C/C++, Java, and Ada). So, the searchable strings in the following line of code are “foreach”, 1, and 10:

```
foreach (i=1, i<10, i++)
```

- You cannot use Instant Search to find strings that cross punctuation boundaries or to search for punctuation itself. For example, you cannot search for “i=1”. You can search for strings that contain spaces (such as text in comments) by surrounding them with quotes.
- You can narrow the search to look within strings, identifiers, and comments. By default, it searches for all three types of matches. For example, the following search finds “test” only in quoted strings:

```
string:test
```

The following search finds “test” only in identifiers such as variable and function names:

```
identifier:test
```

The following search finds “test” only in comments:

```
comment:test
```

- You can use Boolean searches. The default is that multiple search terms are ORed. So, a search of “for delta” is the same as a search of “for OR delta”. Both match files that contain either “for” or “delta”. Remember that the search string is used to match terms in the entire file, not just in a single statement.
- If you want to AND the terms, use a search like “for AND delta”. This matches files that contain both “for” and “delta”.
- You can use the + operator to require that a search term exists in all documents found. For example, the following search finds documents that all contain “delta” and may contain “for”:

```
+delta for
```

- You can use the NOT (or -) operator to remove any documents that contain a particular search term from the results. For example, the following searches find documents that contain “delta” or “delta0” but not “delta2”:

```
delta delta0 NOT delta2
```

```
delta delta0 -delta2
```

- You can use parenthesis to define the order of Boolean operators in searches. For example:

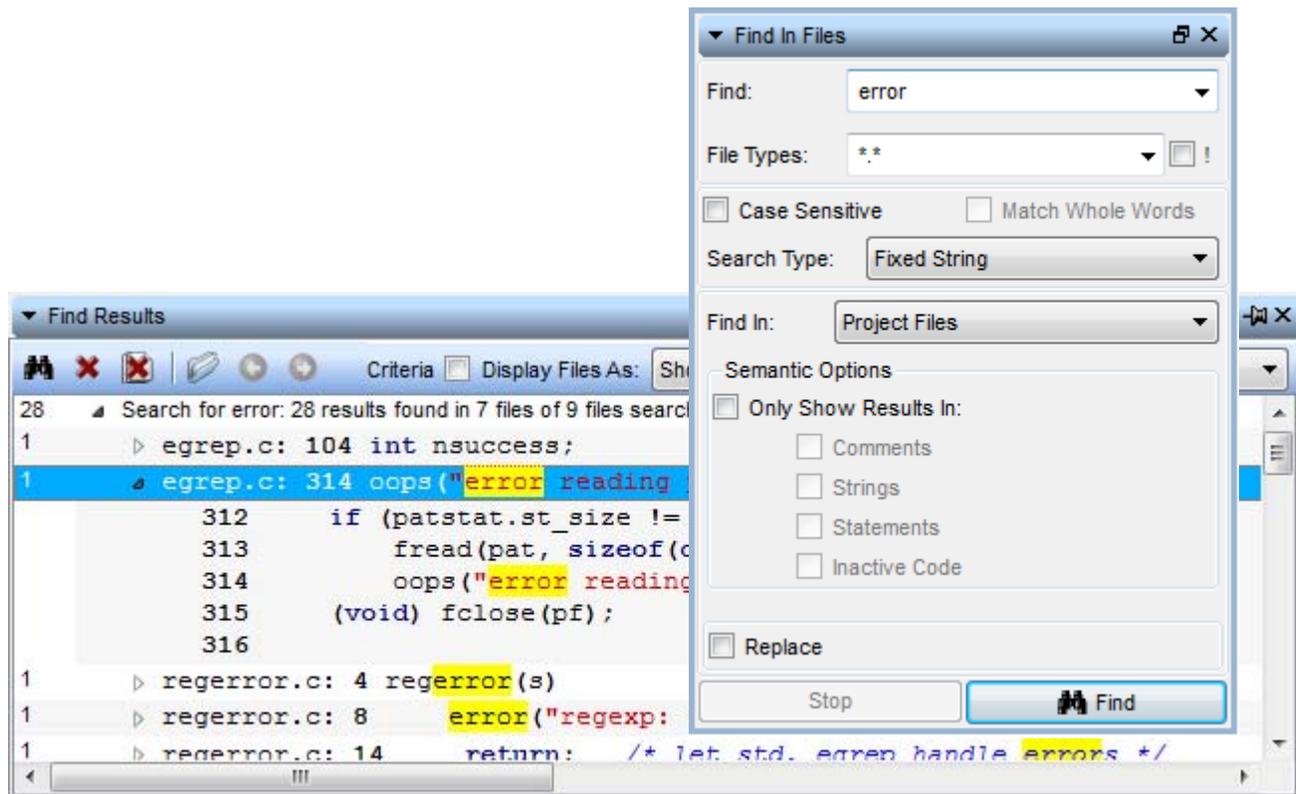
```
(delta0 OR delta1) AND change
```

- You can perform a “fuzzy” search by placing a tilde (~) at the end of a search term. For example boo~ matches foo, too, and book.

Find in Files

You may search all project files or another selection of files for the occurrence of a text string or regular expression. Matches are shown in the *Find Results* window and can be visited in the source code by double-clicking on any line in the results. You can switch between the Find in Files and Replace in Files (see page 146) dialogs by checking the **Replace** box.

To open the Find in Files tool, choose **Search > Find in Files** from the menu bar, choose **Find in...** from any context menu, or press F5.

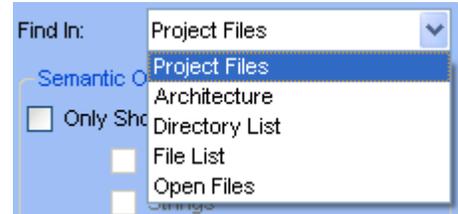


The Find in Files area allows you to search multiple files for the occurrence of a string. In previous versions, this feature was called Hyper Grep for its similarity to the UNIX command *grep*. Specify a search as follows:

- **Find:** Type the string for which you want to search. The other fields control how this search is performed. The drop-down list lets you select from recent Find strings.
- **File Types:** You can select file extensions for various languages to narrow the search. Or, type your own file extension pattern. Leave this field blank to search all files. You cannot use this field if you have the **Find In** field set to “Open Files”. Check the “!” box to the right of the **File Types** field to exclude the selected file types from the search and search all other files in the project.

- **Case Sensitive:** Check this box for case-sensitive searching. The default is to ignore case.
- **Match Whole Words:** Check this box to match whole words only in regular expressions (“test” matches “test” but not “testing”). For fixed string and wildcard searches, word boundaries are ignored.
- **Search Type:** Choose whether to use Fixed String, Wildcard, or Regular Expression matching. See page 150 for details.

- **Find In:** Choose whether to search project files (either all files or just the open files), files in architecture nodes you select, files in directories you select, or files you select.



If you select “Architecture” here, you can click + to browse for an architecture node. If you select “Directory List” or “File List”, you can click + to browse for directories or files and can sort the list with up and down arrows. If you select “Open Files”, all files that are currently open are searched.

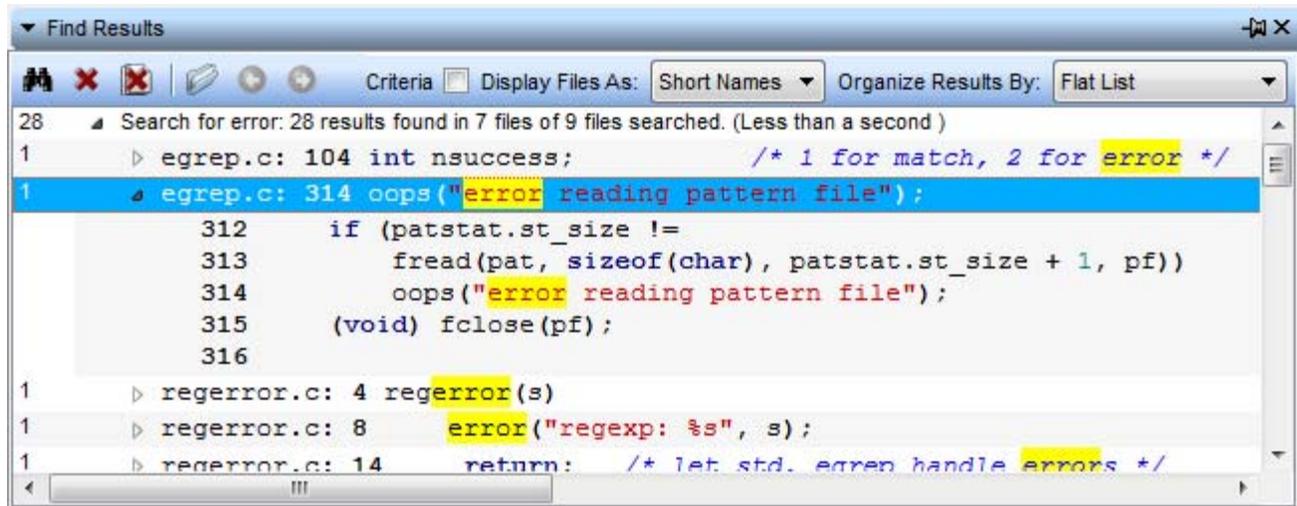
When you right-click on an entity in source code or elsewhere, the **Find In** command lets you choose one of these options for the selected text string. The Find and Find In fields are filled in for you automatically.

- **Semantic Options:** If you choose to Find In “Project Files” or “Architecture”, you can check the **Only Show Results In** box to be able to control which matches are reported. Then you can check any combination of the **Comments**, **Strings**, **Statements**, and **Inactive Code** boxes to include those types of lines in the results. You must check at least one of these boxes if you check the **Only Show Results In** box.
- **Replace:** Switch to the Replace in Files (see page 146) dialog by checking this box.

Click **Find** after specifying the search criteria. A list of all matching instances will be displayed in the *Find Results* window. If the search is taking a long time and you want to change the criteria, you can click **Stop**.

Find Results

The Find Results window lists the matches found. Each line where the string occurs is listed in the Results list.



You can view the source code for a match by double-clicking on a result. This opens the Source Editor and highlights the match. See *User Interface > Windows Category* on page 99 for ways to customize the Find Results display.

Multiple searches are shown in the results list. Right-click on the background of the window and choose **Expand All** to expand all search nodes in the window. Or, choose **Collapse All** to compress the list to just the top-level search listing.

The toolbar (and context menu) for the results provides the following controls:

- Search within the currently selected set of results (using the same search bar described in *Searching the Information Browser* on page 127).
- Delete the current set of results.
- Delete all the results.
- Open the selected match in the Source Editor.
- Move to the previous or next match.
- Check the **Criteria** box to show the settings used to perform the search.
- By default, only the name of files is shown. To show full file paths, select “Long Names” from the **Display Files As** drop-down.
- Change the organization of the most recent results by using the **Organize Results By** drop-down. The choices are a flat list (the default), a file-based list, and hierarchies using the architectures.

From the context menu, you can choose **Copy** or **Copy All** to copy the contents of the window as text for pasting elsewhere.

You can reopen the Find Results window by choosing **Search > Show Find Results**. All the results from the current session are shown unless you have used the toolbar in the Find Results window to delete some results.

Replace in Files

You can use the Replace in Files tool by choosing **Search > Replace in Files** from the menu bar or by checking the **Replace** box in the Find in Files tool.

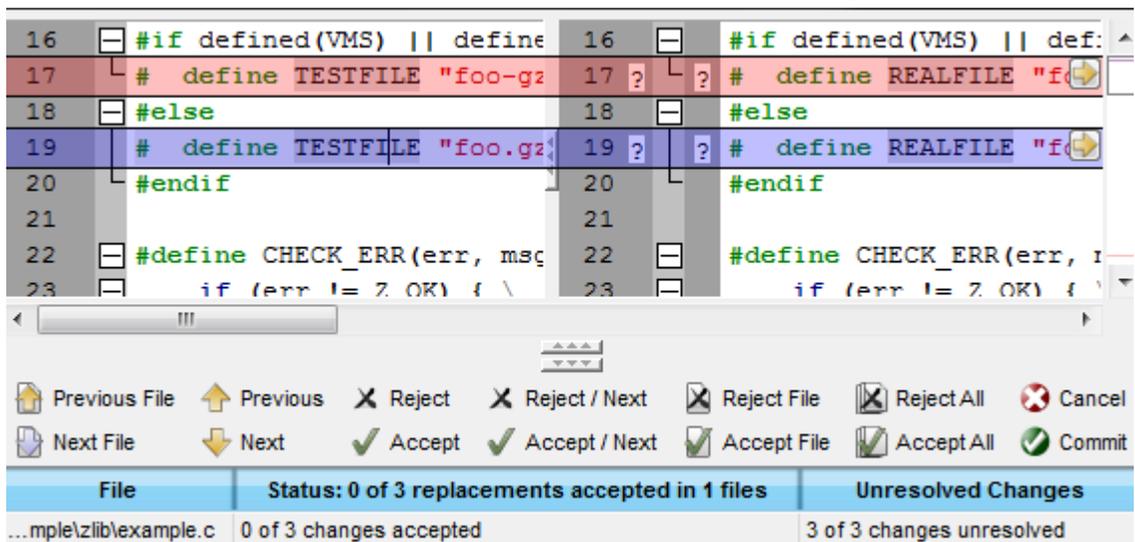
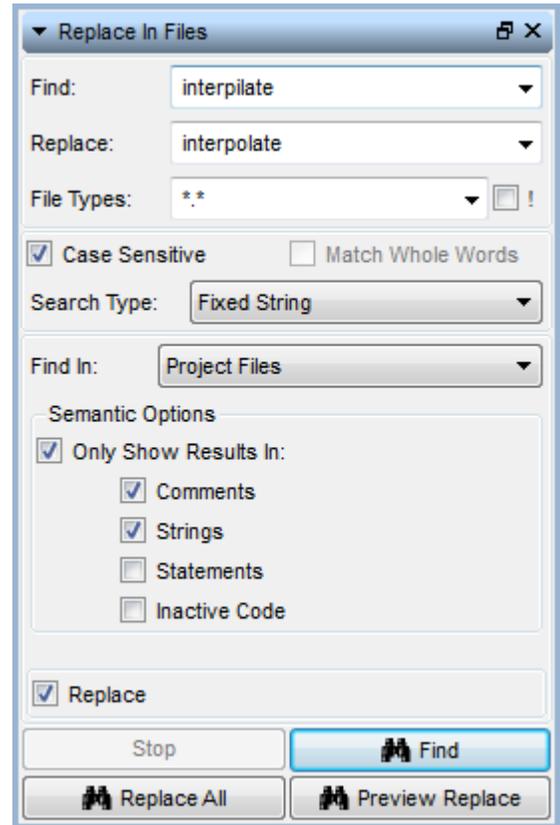
The fields in this tool are the same as those in the Find in Files tool with the following exceptions:

- There is a **Replace** field where you type the text you want to replace the matched string.
- In the **Search Type** field, you can select “Regular Expression - Multiline,” which lets your regular expressions match strings that cross line boundaries.

Understand checks for any unsaved source files. If there are unsaved files, you must click **Yes** to save all unsaved changes before making or previewing the changes.

If you click **Replace All**, you are asked if you want to replace all results automatically. The changes will be saved automatically, so you should be sure you want to make all the changes.

If you click **Preview Replace**, you see the Preview Replace Changes window. You can use this window to accept or reject replacements on a change-by-change basis, file-by-file basis, or all at once.



The top area shows the pre-change code on the left and the post-change code on the right. Replacements are in pink and the currently selected replacement is highlighted in blue. The left side has the **Hide Common Lines** option set so that most lines that will not be affected by the replacement are hidden.

The middle area shows the replacements in patch file format. Such patch files can be used with the Unix patch tool and other similar programs. You can hide this area by clicking the small  fold icon above the area.

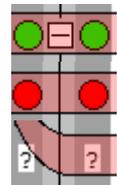
The lower area lists the files where replacements will be made and the number of replacements accepted and unresolved.

The navigation icons let you move to the next and previous file and the next and previous replacement.

The accept and reject icons let you accept or reject replacements on a change-by-change basis, file-by-file basis, or all at once.



Replacements that you have accepted are marked in the source display with green circles. Replacements that you have rejected are marked with red circles. Unresolved replacements are marked with question marks. You can click on a green circle to change it to red, and vice versa.



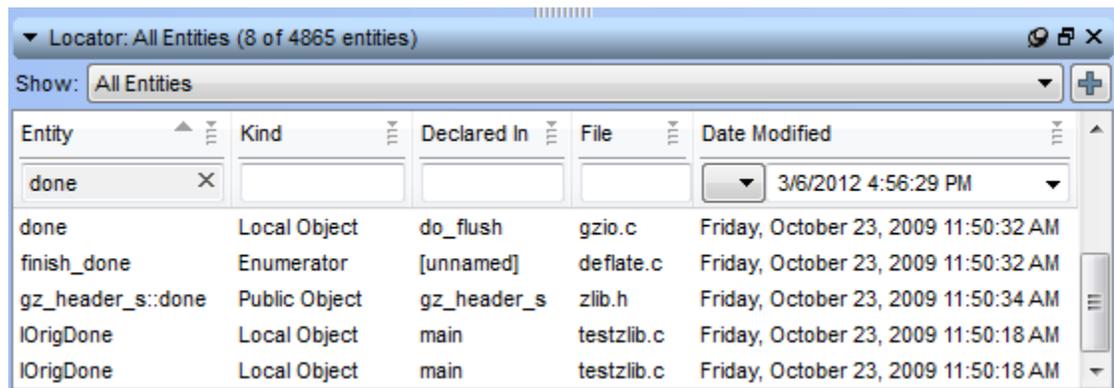
When you have finished resolving differences by either accepting them or rejecting them, click **Commit**. You are asked whether you are sure you want to make the replacement. The message shows how many replacements will be made.

If you decide not to make changes, you can click **Cancel** at any time. If you have accepted any replacements, you see a message that asks if you are sure you want to cancel without making replacements.

Entity Locator

Not all entities fall into one of the tab categories shown in the *Entity Filter*. You can find and learn more about any entity by using the *Entity Locator*, which provides a filterable list of entities in the database. You can filter by name, by entity type, by where the entity is declared, within what container the entity is declared, or when the entity was last modified. You can also use architecture hierarchies to sort entities.

To open the *Entity Locator*, choose **Search > Find Entity** or **View > Entity Locator** from the main menu bar.



As in other windows in *Understand*, when you right-click on an entity anywhere in the *Entity Locator*, a menu of commands available for the item appears.

Resizing Columns

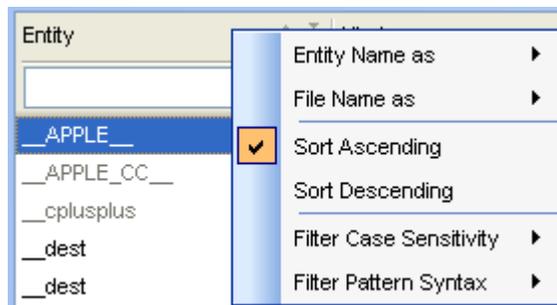
Column widths can be sized to adjust how much of each column is visible. You can drag the column header divider between two columns to resize the column to the left. Or, double-click on the column header divider while the double-headed arrow is displayed and the field to the left of the divider will be expanded or shrunk to the maximum size needed to view all items in that column.

Long versus Short Names

In the *Entity*, *Declared In* and *File* columns, you can right-click the column header or click the drop-down icon  to specify the display format for entity names and filenames. For entities, you can choose the short or full name (which includes the name of the compilation unit). For filenames, you can choose the short, full, or relative path.

Column Headers

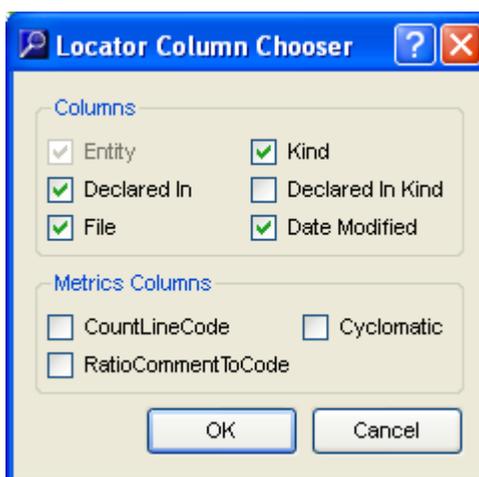
Column headers are tools in the *Entity Locator*. Left-click them to sort according to that column. Right-click a column or click the drop-down icon to see a menu that lets you control how entities are listed, sorted, and filtered.



The entity list may be sorted by any column. Left-click on the column header to toggle between sorting in ascending order and descending order. The default sorting order is in ascending order of entity names.

Choosing Columns

Click the large + icon in the upper-right of the Entity Locator to see the Locator Column Chooser.



The **Entity** column must always be displayed. You can enable or disable the other columns.

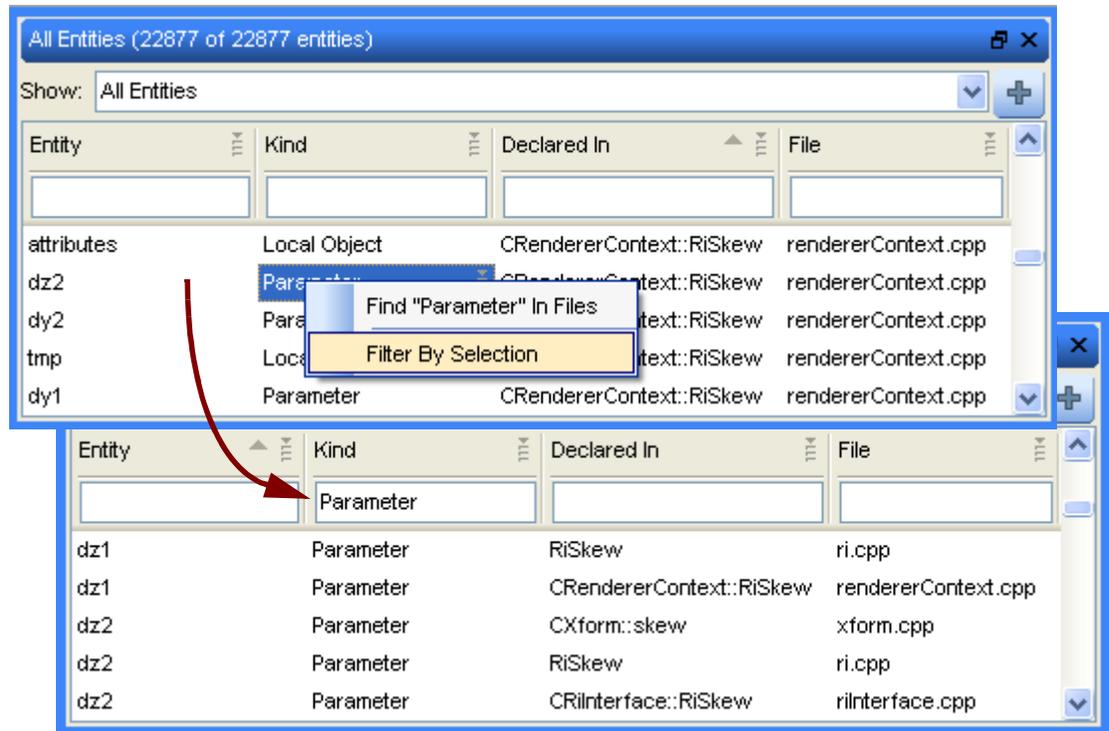
Filtering the List

The field below each column heading lets you filter the entities shown by the *Entity Locator*. The filter can be entered manually or automatically based on what was right-clicked on.

For example, you may filter by the *Kind* column by right-clicking on any item listed in the *Kind* column and selecting **Filter By Selection** from the menu. This filters the list of entities to contain only entities of the kind you selected. The title bar shows how many entities match the filter.

Or, you can simply type a filter in one of the fields. To clear a filter, just delete the text from the field in the column heading.

The following example shows **Filter By Selection** for an entity *Kind*:



To filter the Date Modified column, the left drop-down lets you select a comparison operator (<, <=, =, >=, >), and the right drop-down lets you select a date from a calendar. You can modify the time by typing. You must select a comparison operator in addition to a date in order to filter the entities.

Similarly, the metrics columns allow you to filter with a comparison operator. For example, you can filter the entities to show only those with a Cyclomatic complexity greater than some value or a Comment-to-Code ratio less than some value.

Right-click a column or click the drop-down icon to see the context menu for that column. You can choose for the filter case sensitivity to be **Case Sensitive** or **Case Insensitive** (the default). You can also choose for the filter pattern matching syntax to use fixed strings (the default), wildcards, or regular expressions.

- **Fixed string:** The string you type matches if that exact string is found anywhere in the column value.
- **Wildcard:** These are * or ?, where * matches any string of any length and ? matches a single character. For example, ??ext_io matches any name having 8 letters and ending in *ext_io*.
- **Regular expression:** A powerful and precise way to filter and manipulate text. You cannot use the **Case Sensitive** option if you are using regular expressions.

The following table lists some special characters used in regular expressions.

Symbol	Description	Example
<code>^</code>	Match at the beginning of a line only.	<code>^word</code> Finds lines with word starting in the first column.
<code>\$</code>	Match at end of a line only.	<code>word\$</code> Finds lines that end with "word" (no white space follows word).
<code>\<</code>	Match at beginning of word only.	<code>\<word</code> Finds wordless and wordly but not fullword or awordinthemiddle.
<code>\></code>	Match at end of word only.	<code>\>word</code> Finds keyword and sword but not wordless or awordinthemiddle.
<code>.</code>	A period matches any single character.	<code>w.rd</code> Finds lines containing word, ward, w3rd, forward, and so on, anywhere on the line.
<code>*</code>	Asterisk matches zero or more occurrences of the previous character or expression.	<code>word*</code> Finds word, wor, work, and so on.
<code>+</code>	Match one or more occurrences of the previous character or expression.	<code>wor+d</code> Finds word, worrd, worrrd, and so on.
<code>?</code>	Match zero or one occurrences of the previous character or expression.	<code>wor?d</code> Finds word and wod.
<code>[]</code>	Match any one of the characters in brackets but no others.	<code>[AZ]</code> Finds any line that contains A or Z. <code>[Kk][eE][Nn]</code> Finds any variation of case when spelling "Ken" or "KEen" or "keN".
<code>[^]</code>	Match any character except those inside the brackets.	<code>[^AZ]</code> Finds any line that does not contain the letters A or Z.
<code>[-]</code>	Match a range of characters.	<code>[A..Z]</code> Finds any line containing letters A through Z on them but not lower case letters
<code> </code>	A vertical bar acts as an OR to combine two alternatives into a single expression.	<code>word let+er</code> Finds word, leter, letter, lettter, and so on.
<code>\</code>	Make a regular-expression symbol a literal character.	<code>*\$</code> Allows searching for *. This example finds all lines ending in */

A full explanation of regular expressions is beyond the scope of this manual. UNIX users may refer to the manual page for *regex* using the command “*man -k regex*”. For a comprehensive explanation of *regex* expressions we refer you to the book “*Mastering Regular Expressions*”, published by O’Reilly and Associates (www.ora.com/catalog/regex).

Finding Windows

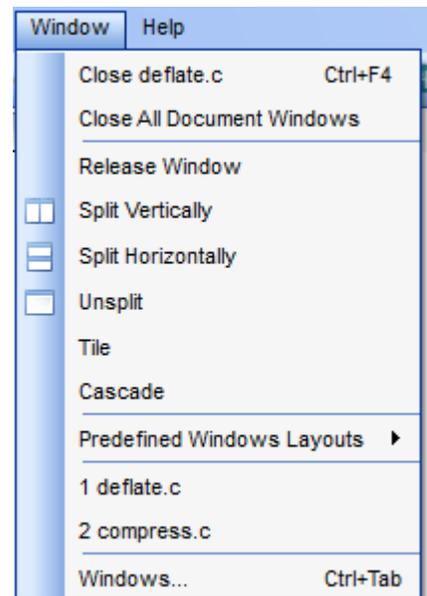
If you have a number of windows open, you can use the options in the **Window** and **View** menus to organize or find particular windows.

You can close the current document window by choosing **Window > Close <current_window>**. You can close all source files, graphical views, and other document windows by choosing **Window > Close All Document Windows**. If you have many windows open in the document area, you can right-click on the tab for the window you are using and choose **Close All**, **Close All But This**, **Close All Tabs to the Left**, or **Close All Tabs to the Right**.

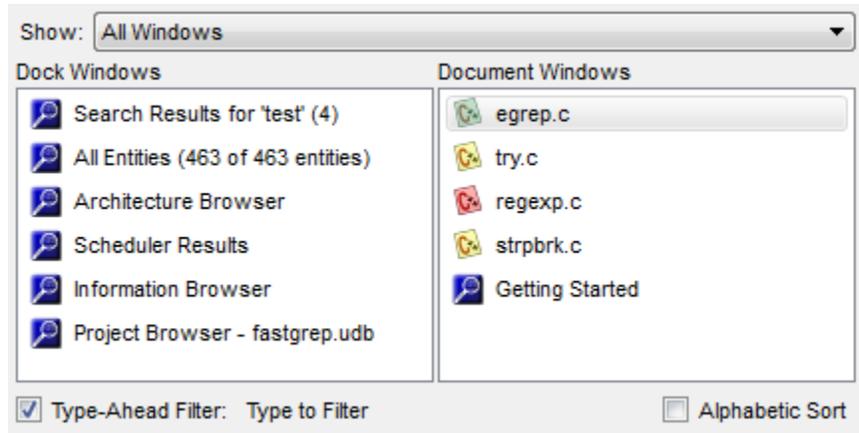
If you choose **Window > Release Window**, the tabbed area changes to a separate window that can be moved around your screen.

The Window menu also lets you use **Window > Split Vertically** or **Window > Split Horizontally** to split the document area. When the document area is split, new areas open in the half that has its box checked. You can drag tabs from one half of the document area to the other as needed. Choose **Window > Unsplit** to remove the split. You can use **Window > Tile** or **Window > Cascade** to arrange the open windows.

The **Window > Predefined Window Layouts** command lets you choose from several standard layouts for common tools. The layouts include the “Tight” layout, “Classic” layout, and “Multi-monitor” layout.



The **Window > Windows** command (Ctrl+Tab) opens a temporary list of currently open windows. When you double-click on an item in this list, the list goes away and focus is given to the item you chose. You can dismiss this area without choosing a window by pressing Esc.



You can reduce the number of windows listed here by choosing a window type from the **Show** list. Or, you can check the **Type-Ahead Filter** box and begin typing some characters in the name of the window you are looking for. Checking the **Alphabetic Sort** box sorts both the docked windows and the document windows. The list of windows is filtered to match the string you type after checking the box.

Source Visiting History

You can move forward or backward through the history of your source code visiting locations using **Previous** and **Next** icons in the toolbar. This history is stored even between *Understand* sessions.



You can click the down-arrows to see the full list of source locations in the history.

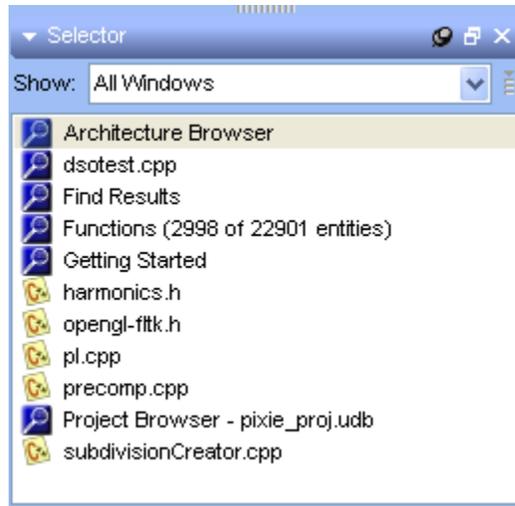
The source locations are stored as line numbers, not by entity name. If you want to save locations by entity rather than line number, see *Favorites* on page 135.

View Menu Commands

If you have parsed the project during this session, you can use the **View > Last Parse Log** command to reopen the log.

The **View > Window Selector** command opens an area that lists currently open windows. By default, this area lists only document windows, but you can use the **Show** drop-down to change the type of window listed.

Click a window name to make it active. By default, the Selector lists all windows, but you can choose to show only Editor windows or various other window types. The icons indicate the type of window, including whether the source file is unsaved.



You can use the drop-down icon  to change the order from alphabetic to most recently used. Click an item in this list to give that item focus.

When the Selector area is active, you can type a filter to quickly narrow the list. The current filter is shown at the bottom of the Selector area. Press Backspace to erase the filter.

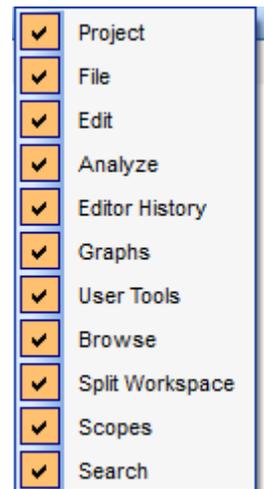
Using the Selector is a convenient way to perform actions—such as Close—on multiple windows by selecting multiple windows from the list, right-clicking, and choosing **Close Selected Window(s)** or **Close Unselected Window(s)**.

If you have created bookmarks in your source code (page 166), you can use the **Bookmarks** command in the View menu to open the list of bookmarks.

Displaying Toolbars

You can hide or display categories of toolbar icons by right-clicking on the toolbar or menu bar and choosing a category. The toolbar is separated into the following categories: Project, File, Edit, Analyze, Editor History, Graphs, User Tools, Browse, Split Workspace, Scopes, and Search.

You can also hide and display toolbar sections by choosing **View > Toolbars** from the menus.

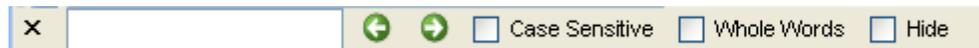


Searching in a File

The search techniques described in this section are used to search a single source file.

Find Next and Previous

To search quickly within the current file, press Ctrl+F (or choose **Search > Find**). The status bar of the Source Editor changes to a search bar.

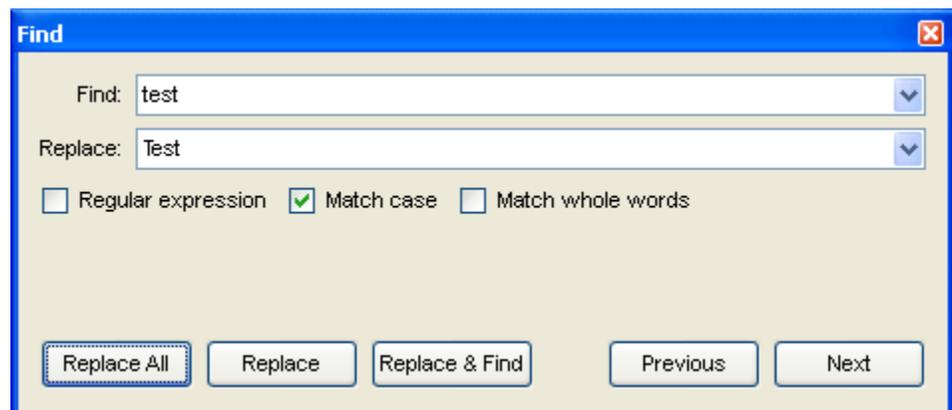


You can type a string in the field. As you type, matches for that string are highlighted in the Source Editor. Click **Previous** or **Next** to move from match to match. You can also check the **Match Case** and **Match Whole Words** boxes to modify how the search is performed.

If you check the **Hide** box, then as soon as you click on the code, the incremental search bar is hidden. When you press Ctrl+F again, your last search is shown. Use Ctrl+Shift+F to find the previous occurrence.

Find & Replace

If you want to use Search-and-Replace or regular expressions for searching, you can use the Find dialog. To open this dialog, choose the **Search > Find & Replace** menu item or press Ctrl+Alt+F.



In the **Find** field, type the string you want to find.

You can check the **Regular expression**, **Match case**, and **Match whole words** boxes to modify how the search is performed. If you check the **Regular expression** box, you can use UNIX-style pattern matching. For a list of some of the capabilities of regular expressions, see page 149.

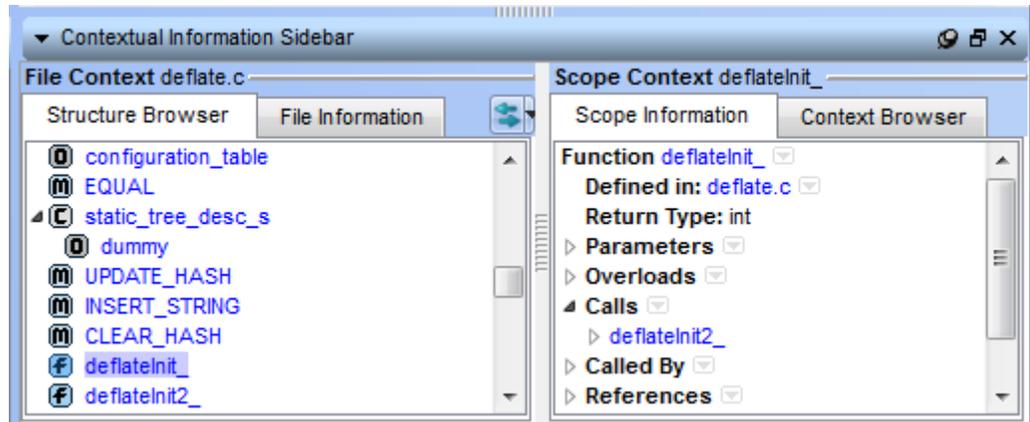
If you want to replace the string you are finding, type that in the **Replace** field.

Click **Previous** or **Next** to search in either direction. Click **Replace All**, **Replace**, or **Replace & Find** if you want to replace the string that was found.

The Find dialog searches only individual files. To search multiple files, see *Find in Files* on page 143.

Contextual Information Sidebar

The Contextual Information Sidebar (CIS) is similar to the Scope List (see page 159), but more powerful. You can open the CIS by choosing **View > Contextual Information** from the menus or clicking the  icon in the toolbar.



The CIS shows the structure and information for the currently active Source Editor. The tabs in the CIS provide the following information:

- **Structure Browser:** This is an expanded scope list for the current file. It lists the names of structures in the file. In addition to functions, it lists includes, macros, classes, and more. The icon next to the name indicates the type of entity. If you point your mouse cursor at an item, the hover text shows the entity type and name. Press Ctrl+F to search within this tab.
- **File Information:** This tab provides an Information Browser for the current file.
- **Scope Information:** This tab provides an Information Browser for the current entity—that is, the one highlighted in the Structure Browser tab.
- **Context Browser:** This tab shows the current entity's location in the hierarchy on the left and the entities it contains on the right.

The  switch icon (Ctrl+,) to the right of the File Information tab changes the current file in the Source Editor and the CIS to a file in the same directory with the same name but a different file extension (the “companion file” if such a file exists). For example, the switch icon can toggle from a .c or .cpp file to a .h file with the same name.

As always, right-clicking in any of these tabs provides links to more information about each entity.

Chapter 6

Editing Your Source

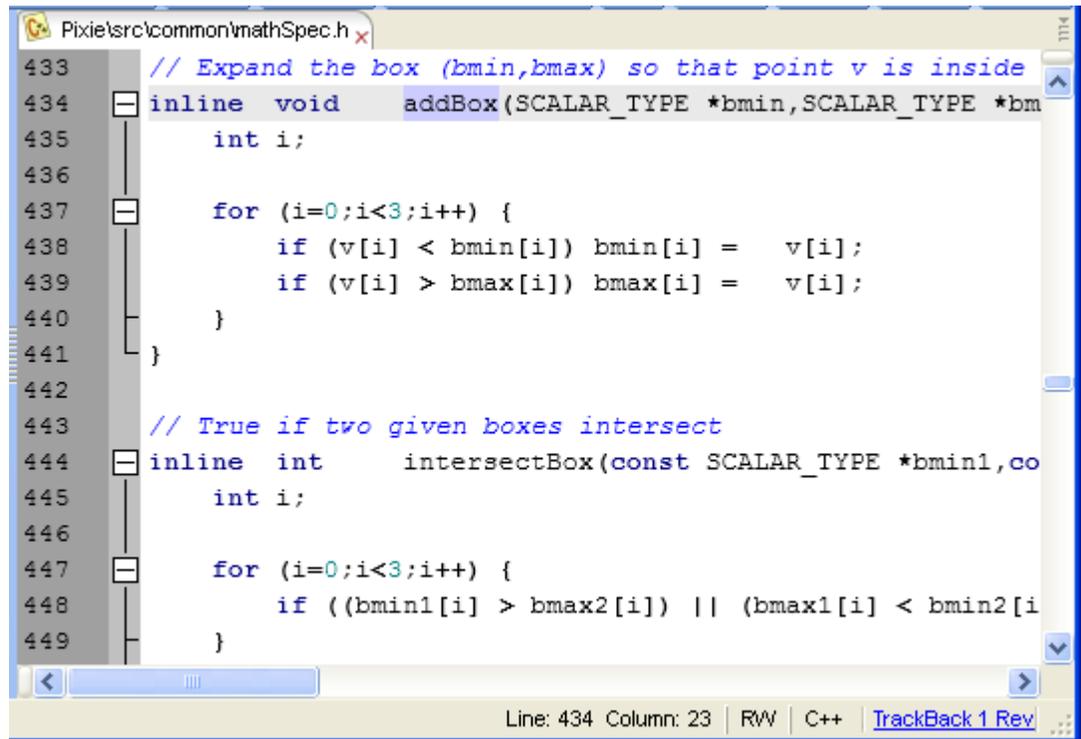
This chapter covers *Understand's* source and text file editor.

This chapter contains the following sections:

Section	Page
Source Editor	158
Saving Source Code	162
Other Editing Features	163
Annotations	168
Printing Source Views	174

Source Editor

The **Source Editor** offers a full featured source code editor, with syntax coloring and right-click access to information most entities in your code.



```

Pixie\src\common\mathSpec.h x
433 // Expand the box (bmin,bmax) so that point v is inside
434 inline void addBox(SCALAR_TYPE *bmin,SCALAR_TYPE *bmax,
435 int i;
436
437 for (i=0;i<3;i++) {
438     if (v[i] < bmin[i]) bmin[i] = v[i];
439     if (v[i] > bmax[i]) bmax[i] = v[i];
440 }
441 }
442
443 // True if two given boxes intersect
444 inline int intersectBox(const SCALAR_TYPE *bmin1,co
445 int i;
446
447 for (i=0;i<3;i++) {
448     if ((bmin1[i] > bmax2[i]) || (bmax1[i] < bmin2[i]
449 )

```

Line: 434 Column: 23 | RW | C++ | [TrackBack](#) [1 Rev](#)

The line numbers and “fold” markings to expand/collapse blocks of code can be turned on and off in the **Editor** category of the Understand Options dialog you can open with the **Tools > Options** command (see page 105). The display font and a number of other items can also be changed in the **Editor** category. You can also enable bookmarks, indent guide marking, and a right margin marker (page guide) in that category of the dialog.

You can zoom in or out to make the text larger or smaller by choosing one of the **View > Zoom** menu options.

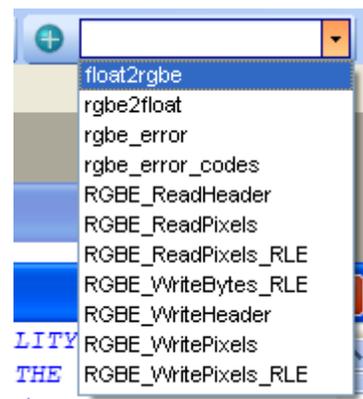
The **Editor > Styles** category of the Understand Options dialog (see page 110) lets you change the colors used for different types of source code. The **Key Bindings** category (see page 100) shows a list (and lets you modify the list) of keystrokes you can use in the Editor.

Scope List

You can jump to a particular function, procedure, or other language-specific construct in the current source file by selecting from the scope drop-down list in the toolbar. The drop-down list shows all such constructs in the file the last time the project was analyzed.

You can click the + icon to move the list to a Scope tab in the area where the Entity Filter is shown. This tab lists constructs in the current source file. This tab is useful for jumping around in large files.

You can right-click on the Scope List area to choose a sort order from the context menu. The ascending and descending orders sort alphabetically or reverse alphabetically. The default is file order.



If you choose **View > Scope List**, you see an area that lists the functions, procedures, and other language-specific constructs in the current source file. The numbers next to each name are the line numbers where the entity is declared in the file. Single-click on an item to view information about it in the Information Browser. Double-click on an item to jump to the location where that item is declared or created and to highlight all occurrences of that name in the current source file.

For more power than the scope list, use the *Contextual Information Sidebar* on page 156.

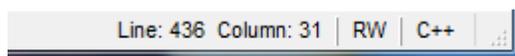
Status Icons

Each file in a Source Editor has a status icon in its upper-left title bar. The letter in the icon indicates the type of file. The icon color indicates whether the file has been modified but not yet analyzed. An asterisk by the filename means the file has unsaved changes.

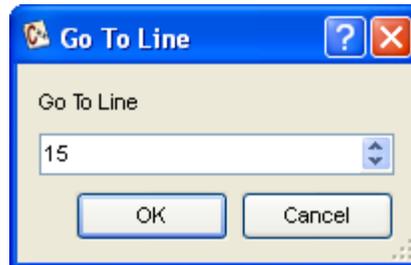
-  Yellow icon = parsed project file (has not been modified)
-  Red icon = modified project file (needs to be parsed)
-  White icon = file not in the project

Status Line

When a Source Editor is the active window, the status bar at the bottom of the *Understand* window shows the current line number and column number, whether the file is in read-write or read-only mode, and the source language at a quick glance.



If you click the line number in the status bar (or choose **Search > Go to Line** from the menus), you can use the Go To Line dialog.



If you click the RW (read-write) indicator, it changes the mode to RO (read-only).

If you click the language, you can choose which language *Understand* should treat this file as.

Selecting and Copying Text

Text can be selected (marked) then cut or copied into the Windows (or X11) clipboard. Selecting text works as standard for the operating system in use. On Windows, dragging while holding down the left mouse selects text. Alternately you can hold down the Shift key and move the cursor (via arrows or the mouse). Choose the **Select All** command in the **Edit** menu or the context menu to select the entire file.

If you hold down the Alt key (Ctrl key on X Windows), you can select a rectangular area of source code—for example, to exclude tabs in the left margin from the copied text. You can also paste rectangular areas of code within the Source Editor.

Once you select text, you can use the **Cut** and **Copy** commands in the **Edit** menu or the context menu. You may then paste the text into other applications as needed.

Browse Mode

You can switch a Source Editor to “Browse” mode by clicking the **Browse** button in the main toolbar or choosing **View > Browse Mode** from the menus. When you are in Browse mode, the icon is highlighted.



When you are in Browse Mode, entities in the code act as links. An underline is shown when your mouse cursor moves to a link. Clicking a link moves you to the declaration of that entity and updates the Information Browser to show details about that entity.

If the declaration of an entity you click on is not found, a message is shown in the status bar and your computer beeps.

When you are in Browse Mode, you can still edit the file and the keyboard and right-click function the same as in regular mode. Only left-clicking the mouse is different.

You can temporarily enter Browse Mode by holding down the Ctrl key while using a Source Editor window. You can toggle Browse mode by pressing Ctrl+Alt+B.

See page 111 for settings to control the behavior of Browse Mode.

Context Menu

The context menu in the Source Editor provides access to a number of exploration and editing features. Many of them let you find specific information about the entity you right-click on.

The following exploration features are typically included in the context menu (depending on where you click):

- View Information (see page 125)
- Graphical Views (see Chapter 10)
- Edit Source/Definition
- User Tools (see page 291)
- Explore (see page 132)
- Find in... (see page 143)
- Add Favorite (see page 135)
- Metrics Charts (see page 220)

The following editing features are also typically included in the context menu:

- Undo / Redo
- Cut / Copy / Paste (see page 160)
- Select All (see page 160)
- Jump to Matching Brace (see page 163)
- Select Block (see page 163)
- Hide/Show Inactive Lines (see page 163)
- Fold All (see page 163)
- Soft Wrap (see page 165)
- Comment Selection / Uncomment Selection (see page 164)
- Change Case (see page 164)
- Revert (see page 162)
- Add Bookmark (see page 166)

Hover Menu

If you point the mouse cursor at an entity in source code, you see a message that shows declaration information about that entity. For example, pointing to a variable shows the variable's type, pointing to a constant shows the constant's value, and pointing to a function call shows the parameters and return value.

```

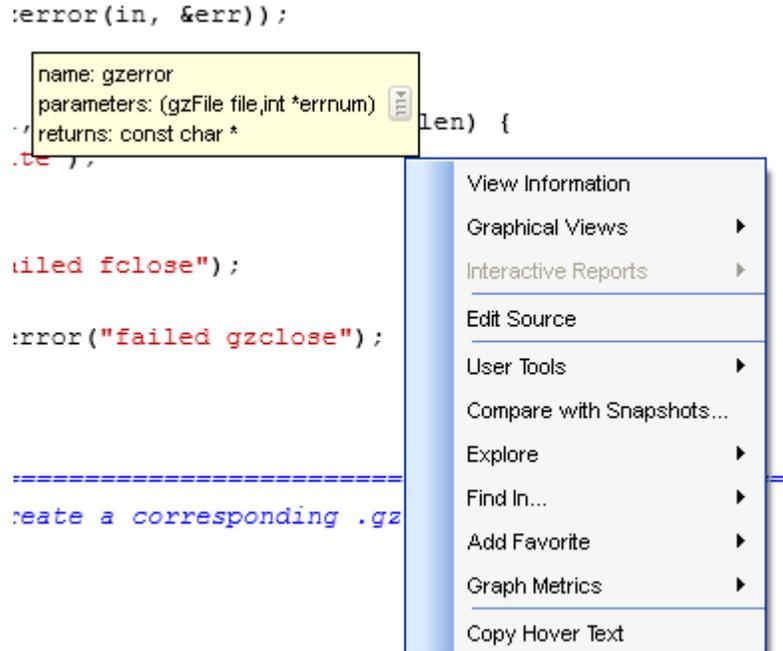
if (len < 0) error (gzerror(in, &err));
if (len == 0) break;

if ((int)fwrite(buf, 1,
                error("failed fwrite");

```



Notice the drop-down menu icon on the message. If you click this icon, you see the “hover menu”, which contains some of the same commands in the context menu for the same entity.



An additional item in the hover menu is **Copy Hover Text**. When you choose this item, the declaration information shown in the hover message is copied to your clipboard, so that you can paste it as needed.

Saving Source Code

If you have edited a source file, you can click , press **Ctrl+S**, or choose **File > Save** to save your changes.

You can choose **File > Save As** to save to a different file. If you save a project file to another filename, you will be asked whether you want to add the new file to the project.

If you have edited multiple source files, you can click  or choose **File > Save All** to save changes to all modified files.

If you want to ignore changes you have made choose since the last save, right-click in a file and choose **Revert**.

You can close the current source file by choosing **Window > Close <current_file>** from the menus. You can also middle-click on the tab above the source file area to close that tab (if your mouse has a middle button).

You can close all source files by choosing **Window > Close All Document Windows**. You can also right-click on the tab for the source file area and choose **Close**, **Close All**, **Close All But This**, or **Close All Tabs to the Right/Left**.

Other Editing Features

The Source Editor also provides several other options for displaying and editing files.

Bracket Matching

A handy feature of the *Understand* editor is syntax bracket matching. Use this feature to find the matching ending character for a brace, parenthesis or bracket. Symbols matched are (), { }, and []. Matching isn't done inside comments.

Pressing Ctrl+j (or right-click and **Jump to Matching Brace**) jumps the editor to the matching end or beginning brace. Ctrl+j isn't active unless your editing cursor is by a symbol that it can match. Another Ctrl+j takes you back where you started. You can also choose **Search > Go to Matching Brace** from the menus.

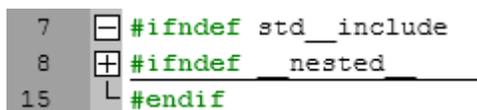
Pressing Ctrl+Shift+J (or right-click and **Select Block**) selects all the text from the bracket to its matching bracket.

Brackets without a match are highlighted in red when you move your cursor to them. Brackets with a match are highlighted in green.

When your cursor is on a preprocessor directive that has a match (for example, #ifdef and #endif), you can use Ctrl+j (or right-click and **Jump to Matching Directive**) to move your editing cursor to the match.

Folding and Hiding

The - and + markings next to the line numbers allow you to "fold" the code to hide blocks such as functions, if statements, and other statements that have a beginning and end.



```

7  - #ifndef std_include
8  + #ifndef nested
15 L #endif

```

If you right-click on the code, you can choose **Fold All** to close all the open blocks. You can also fold and unfold source code by choosing **View > Fold All** from the menus.

You can add explicit fold markers to code in languages where // is treated as the beginning of a comment. For example:

```

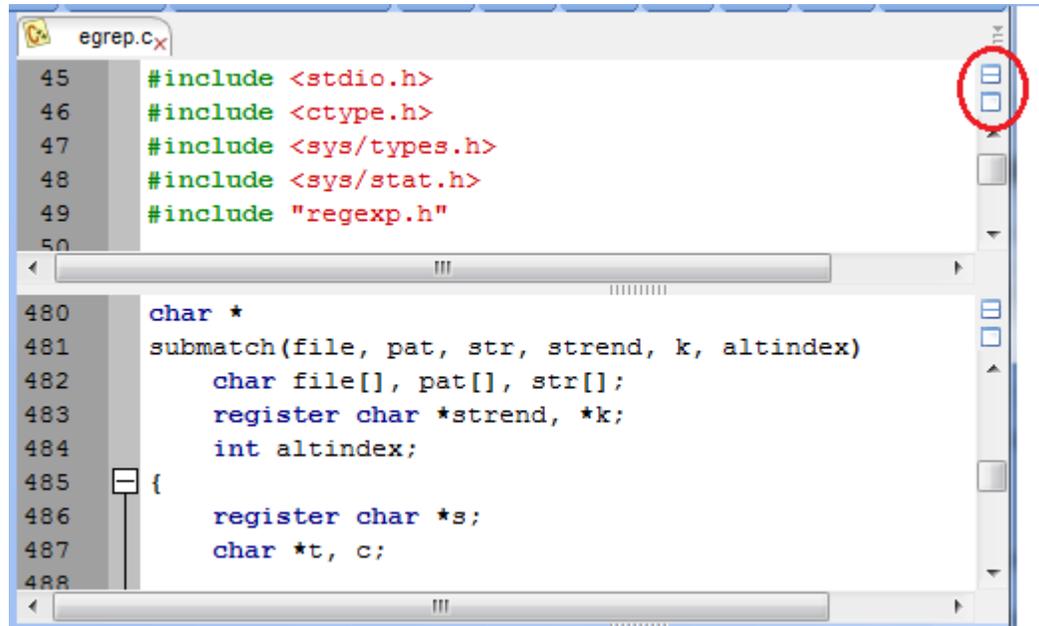
//{{
/* code to hide when folded */
//}}

```

You can also choose **Hide Inactive Lines** to hide preprocessor lines that are not active because a preprocessor macro is not defined. Choose **Show Inactive Lines** to view all lines again. You can also toggle this setting by choosing **View > Hide Inactive Lines** from the menus.

Splitting the Editor Window

You can click the **Split** icon (circled below) to divide the source editor into two or more separately scrollable panes. Click one of the **Join** icons to merge two panes.



Commenting and Uncommenting

You can comment code that you have selected by right-clicking and choosing **Comment Selection**. To remove the comment characters, right-click and choose **Uncomment Selection**. You can do the same thing using the **Edit > Comment Selection** and **Edit > Uncomment Selection** commands in the menus.

Note that nested comments within the selection are not parsed.

Fixing Indentation

You can make the indentation of selected code match standard usage by selecting the code, right-clicking, and choosing **Reindent Selection**. See *Editor > Advanced Category* on page 107 to configure the indentation settings.

Changing Case

You can change the case of selected text in the Source Editor. Follow these steps:

- 1 Select a word or words in the source code.
- 2 Choose **Edit > Change Case** from the menus, or right-click and choose **Change Case** from the context menu.
- 3 Choose the type of case you want to apply to the selection. The choices are as follows:

Choice	Default Keystroke	Original	Result
Lowercase	Ctrl+U	Test_me please	test_me please
Uppercase	Ctrl+Shift+U	Test_me please	TEST_ME PLEASE
Invert Case	Ctrl+Shift+I	Test_me please	tEST_ME PLEASE
Capitalize	Ctrl+Alt+U	Test_me pleaSe	Test_me PleaSe

Line Wrapping

Normally, lines are cut off on the right if your Source Editor window is not wide enough to display the full line length. You can make the Source Editor wrap long lines to the next line if you like. To do this, right-click in the Source Editor and choose **Soft Wrap**. You can also change the wrapping mode by choosing **View > Soft Wrap** from the menus.

The line breaks displayed are for display only; no actual line breaks are added to your source file.

See *Editor > Advanced Category* on page 107 to change the wrap mode for source code printing.

Insert and Overtyping Modes

Normally, text to the right of your typing cursor is shifted as you type. This is called Insert mode. To switch between Insert mode and Overtyping mode, in which text to the right of the cursor is replaced character-by-character as you type, press the **Insert** key or choose **Edit > Toggle Overtyping** from the menus.

Sorting Lines Alphabetically

To sort a group of lines into alphabetical order, select the lines, right-click and choose **Sort Selection**.

Keyboard Commands

To see a list of keystrokes that work in the Source Editor, choose **Tools > Options** and go to the **Key Bindings** category. For example, Ctrl+Alt+K cuts the text from the cursor position to the end of the line. And, Ctrl+T transposes the line at the cursor position with the line above it.

Another way to see a list of key bindings is to choose **Help > Key Bindings**. Search for the line that says "Editor" (around line 110) to get to the beginning of the keystrokes for the Source Editor windows.

Recording and Replaying Macros

You can record and replay a set of editing changes that you want to be able to repeat. These are called macros.

To record a macro, follow these steps:

- 1 Choose **Tools > Editor Macros > Record Macro** from the menus or press Ctrl+Alt+M.
- 2 Perform the steps you want to be able to repeat.
- 3 Choose **Tools > Editor Macros > Record Macro** again or press Ctrl+Alt+M.

To replay a macro, move your cursor to the desired start location and choose **Tools > Editor Macros > Replay Macro** from the menus or press Ctrl+M.

Creating and Opening Files

You can use the Source Editor to create an untitled blank file by choosing **File > New > File** from the menus. You can open files, whether they are in your project or not, by choosing **File > Open > File**.

When you right-click on a filename, the context menu provides options to **Edit File** and to **Edit Companion File**. For example, the companion file of encrypt.c is encrypt.h.

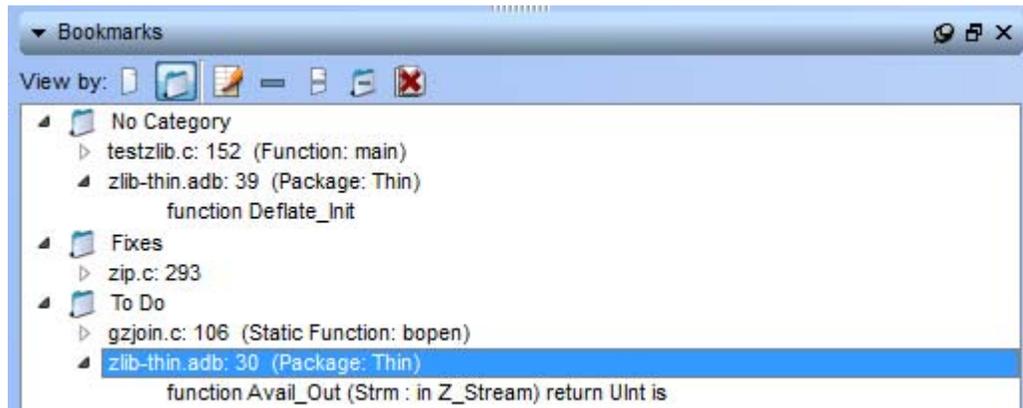
Bookmarking

You can create “bookmarks” in your code by right-clicking on a line and choosing **Add Bookmark** from the context menu. Or choose **Edit > Bookmarks > Toggle Bookmark** from the menus. Lines with a bookmark have a red arrow next to them.

```
5 |▶| #define __nested__
```

In a file with multiple bookmarks, you can right-click and choose **Previous Bookmark** or **Next Bookmark** to quickly move between places in a file. These commands are also available under **Edit > Bookmarks** in the menus.

You can open a Bookmarks area to view a list of all your bookmarks in all your files by choosing **View > Bookmarks** from the menus.



Double-click on a bookmark to move to that location in the Source Editor. If you create a bookmark *inside* an entity, the Bookmarks area shows the name and type of entity that contains the bookmark. For example, the function name is shown if you create the bookmark on the first line of code inside a function.

The toolbar for this area lets you manage your bookmarks in the following ways:

  You can use the **View by** icons to switch between a file-based and a category-based view. The file-base view lets you expand filenames to see the bookmarks in that file. The category-based view lets you assign bookmarks to categories you create.

 Select a bookmark and click this icon to change the category the bookmark is in. To create a new category, type the name and click **OK**. To use an existing category, select it from the list.

 Select a bookmark and click this icon to delete that bookmark.

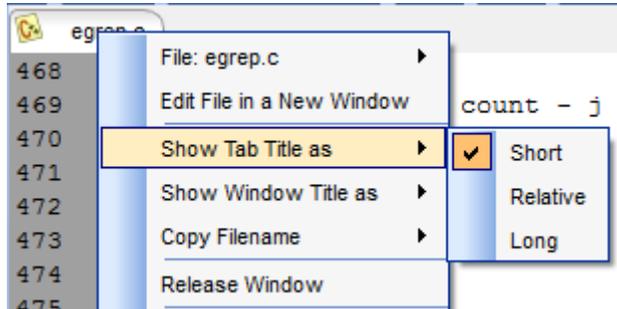
 Select a file in the file-based view and click this icon to delete all the bookmarks in this file. You can also select a bookmark and click this icon to delete all the bookmarks in the file that contains the selected bookmark.

 Select a category in the category-based view and click this icon to delete all the bookmarks in the category. The category itself is not deleted.

 Click this icon to delete all your bookmarks.

Managing Source Editor Tabs

When you right-click on the tab at the top of a Source Editor, some of the commands allow you to control the behavior of the tab.



If you choose **Show Tab Title as**, you can shorten or lengthen the filename in Source Editor tabs. Likewise, if you choose **Show Window Title as**, you can shorten or lengthen the filename in the *Understand* title bar and any separate Source Editor windows. The **Copy Filename** command lets you copy the long, relative, or short filename to the clipboard.

If you choose **Release Window**, the tabbed area changes to a separate window that can be moved around your screen. Click  to change a tab to a window within the *Understand* window.

Changing the Source Code Font Size

You can change the default display font and font size in the **Editor** category of the Options dialog that you open with the **Tools > Options** command (see page 105).

In addition, you can change the display size of the font for an individual source code window by choosing options from the **View > Zoom** submenu. **View > Zoom > Zoom In** makes the font size larger. **View > Zoom > Zoom Out** makes the font size smaller. **View > Zoom > Reset Zoom** changes the font size back to the default.

Annotations

Annotations let you add comments or notes about entities without changing the source code directly. You can view the annotations inline, following the definition of the entity to which they are attached. They can also be seen in hover text wherever the annotated entity is used.

```

295 FILE *pf;
296 struct stat patstat;
    #issue:CQ0034567 Enhance to provide more statistics
    Me - Tuesday, August 21, 2012 11:40:06 AM
297 static char *pat;
298

```

Each annotation can be “tagged” with a key value pair. Such tagging is useful for organizing your notes using keywords, author names, or any other identifier you want to use.

Adding an Annotation

To add an annotation, follow these steps:

- 1 Highlight an entity, such as a variable or function name, anywhere in *Understand*. For example, you can select an entity in source code, in the Information Browser, or in the Entity Filter.
- 2 Right-click on the entity and choose **Annotate** and the entity you want to annotate from the context menu.

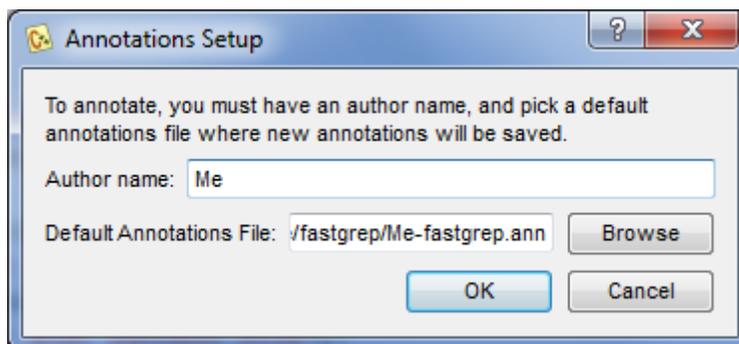
```

3 char * strpbrk(pat1, pat2)
4
5 register char *pat1;
6 char *pat2;
7 {
8
9 register char *cp1,
10
11 cp1 = pat2;
12
13 while (*cp1 != '\0')
14     if ((cp2 = index(
15         return(cp2);
16     cp1++;
17 }

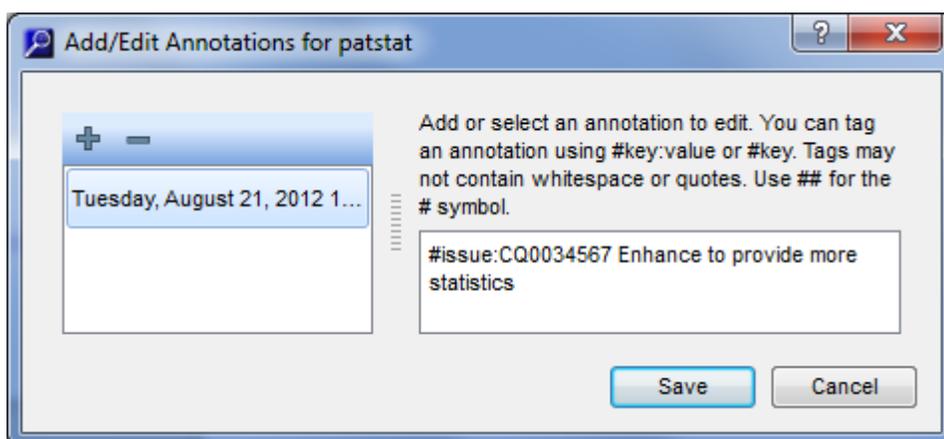
```

- 3 If this is the first time you are adding an annotation, you will see the Annotations Setup dialog. Type your name in the **Author name** field. The annotations are stored in a *.ann file within the project. The **Default Annotations File** is stored in the project directory. The filename includes your name and the name of the project.

(See page 170 to learn about managing annotation files.)



- 4 In the Add/Edit Annotations dialog, click the “+” icon. The current date and time are added to the left box. You can select annotations using this timestamp when you want to edit annotations.



- 5 Type your annotation comment in the right box. If you want to tag your comment so that it will be easy to search for, begin the text with a #key or #key:value tag. For example, you can use #reminder:CodeReview to flag items that should be reviewed. Or, you could use #errorchecks to flag items that need to have their status tested. The author name you entered is automatically associated with your annotations, so you don't need to include your name in a #key:value pair. You can type multiple keys in a single annotation, and keys can occur anywhere within the annotation text. If you want to use a # sign in the annotation text without having it treated as a key, type ##.
- 6 Click **Save**. Your annotation appears in the source code where the selected entity is defined. (See page 170 to control how annotations are displayed.)

You can also add an annotation by clicking on an entity and choosing **Annotations > Annotate > Annotate <entity>** from the menus. Options are shown to annotate the current file, the current entity, and any entities that contain the current entity (such as a function that contains the selected variable).

Editing an Annotation

To edit an existing annotation, follow these steps:

- 1 Highlight an entity that has an annotation anywhere in *Understand*.
- 2 Right-click on the entity and choose **Annotate** and the entity you want to annotate from the context menu.
- 3 In the left box, select the timestamp for the annotation you want to edit.
- 4 Modify the annotation text in the right box.
- 5 Click **Save** to store your changes.

If you edit an annotation that was originally created by someone else, that other person remains the author of the annotation. The timestamp for the annotation is updated to the last time it was edited.

Deleting an Annotation

To delete an existing annotation, follow these steps:

- 1 Highlight an entity that has an annotation anywhere in *Understand*.
- 2 Right-click on the entity and choose **Annotate** and the entity whose annotation you want to delete from the context menu.
- 3 In the left box, select the timestamp for the annotation you want to edit.
- 4 Click the “-” (minus) icon above the list of timestamps.
- 5 Click **Save** to finish deleting the annotation.

Managing Annotation Files and Display

You can choose **Annotations > Annotation Options** from the menus, and then set the following options:

- Your name or username to identify the original author of your annotations.
- The annotation files to look in for this project, and which file is the default for annotations you add. For example, you can have *Understand* display annotations from separate files for everyone working on this project.
- The foreground and background colors to use when displaying annotations from each of the files.
- How to display annotations: inline, as hover text, and with an indicator.

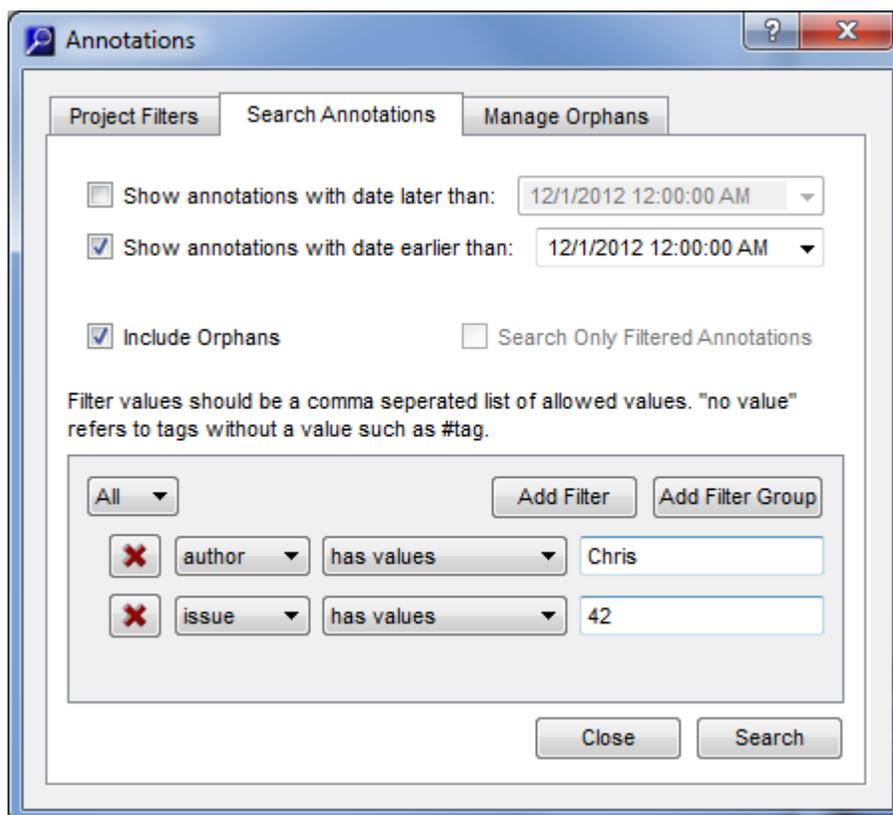
See *Annotations* on page 59 for details about setting these options.

If other developers are also annotating code using *Understand*, choose **Annotations > Refresh Annotations** from the menus when you want to get the latest annotations they have added.

Searching Annotations

You can search for annotations based on the key:value pairs, the author, and the timestamp. To search annotations, follow these steps:

- 1 Choose **Annotations > Search Annotations** from the menus.
- 2 Specify any of the following search parameters you want to use.



- **Date range:** Check one or both date range boxes if you want to find annotations edited after a certain date and time and/or annotations last edited before a certain date and time.
- **Include Orphans:** Check this box if you want the search to also find annotations that are linked to entities that have been deleted. See page 173 for more about orphaned annotations.
- **Search Only Filtered Annotations:** Check this box if you want to limit the search to annotations that match the current filters. See page 172 for more about filtering annotations.
- **Filter values:** You can set up one or more filters for the search based on the author and any #key:value pairs in the annotations. The “has values” and “doesn’t have value” options let you type a value to match or exclude for a #key:value pair. Exact matches for the author name and key values must be used; partial matches and wildcards are not supported. The “any value” option matches any annotation that has that #key, no matter what the value. The “no value” option matches annotations that have that #key, but no #key:value pair. Use **Add Filter** to create another filter, and choose **All** or **Any** to determine how matching is performed.

You can even use **Add Filter Group** to create nested levels of filters that have different settings for **All** and **Any**.

- 3 Click **Search**. The results are shown in the Annotation Search Results area in the main *Understand* window.
- 4 Expand the search results, and double-click on an item to go to the location where that annotation appears in the code. (That is, the location where the entity associated with the annotation is defined.)

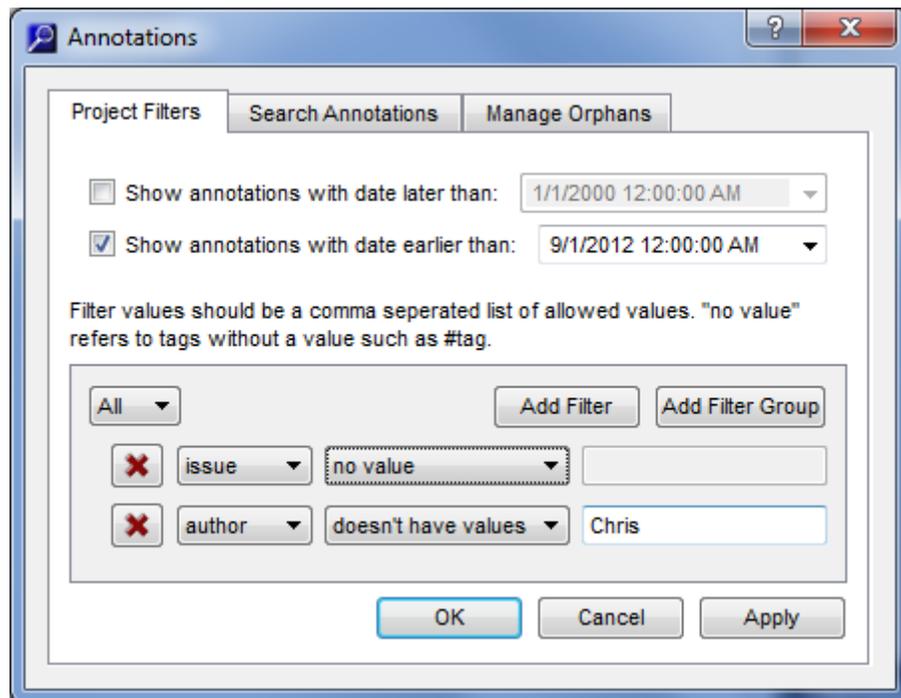
Annotations are stored in *.ann files, which use the SQLite database format. In addition to viewing annotations in *Understand*, you can use other applications that support SQLite to modify and search annotation files.

If other developers are also annotating code using *Understand*, choose **Annotations > Refresh Annotations** from the menus when you want to get the latest annotations they have added.

Filtering Annotations

You can filter annotations based on the key:value pairs, the author, and the timestamp. To filter annotations, follow these steps:

- 1 Choose **Annotations > Filter Annotations** from the menus.
- 2 Specify any of the following filters you want to use.



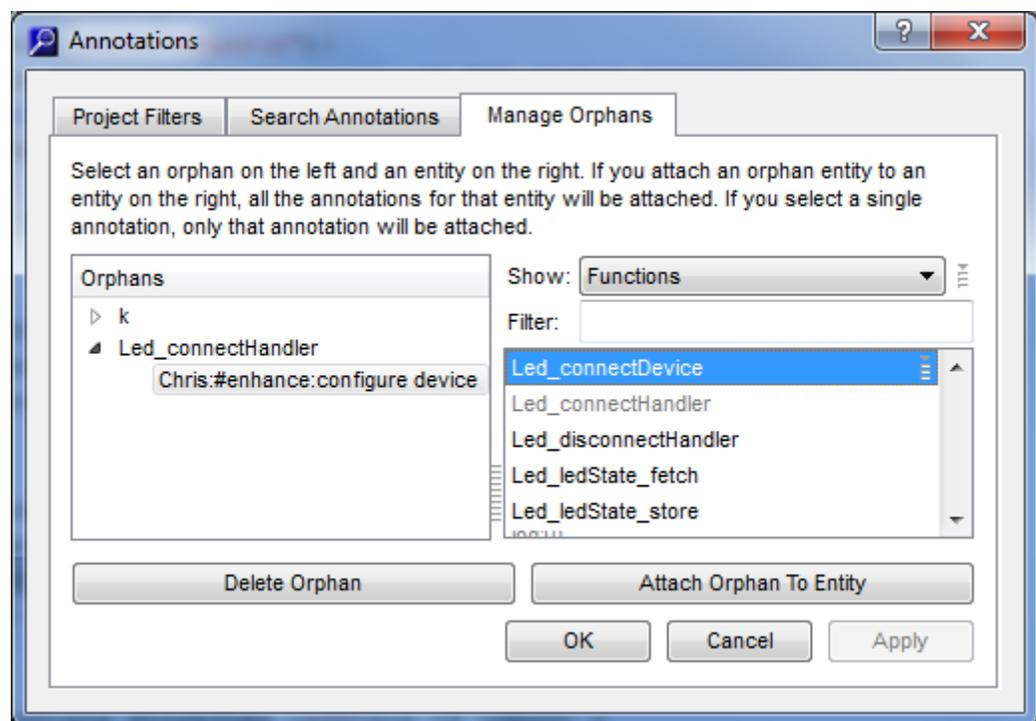
- **Date range:** Check one or both date range boxes if you want to find annotations edited after a certain date and time and/or annotations last edited before a certain date and time.

- **Filter values:** You can set up one or more filters for the search based on the author and any #key:value pairs in the annotations. See page 171 for details on using these fields.
- 3 Click **OK** or **Apply**. The filters you specify are applied to the annotations shown throughout *Understand*.

Managing Orphaned Annotations

If you create an annotation, and later delete the entity with which it was associated, that annotation becomes an “orphan” when you re-analyze the project. Orphan annotations aren’t shown in the code anywhere. You can manage orphan annotations by choosing whether to delete or re-attach them. To manage orphan annotations, follow these steps:

- 1 Choose **Annotations > Manage Annotations** from the menus.



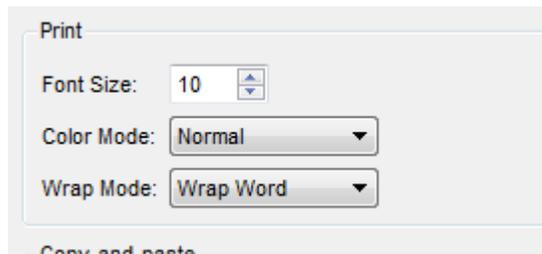
- 2 Expand an orphan in the list on the left to see the annotation text.
- 3 If you want to delete the selected annotation, click **Delete Orphan**.
- 4 If you want to attach the selected annotation to a different entity, select an entity from the list on the right. (You can shorten the list by selecting a type of entity from the **Show** drop-down.)
- 5 Click **Attach Orphan To Entity** to connect the selected orphan to the selected entity. The annotation will be shown in the code where the new entity is defined.
- 6 Click **OK**.

Printing Source Views

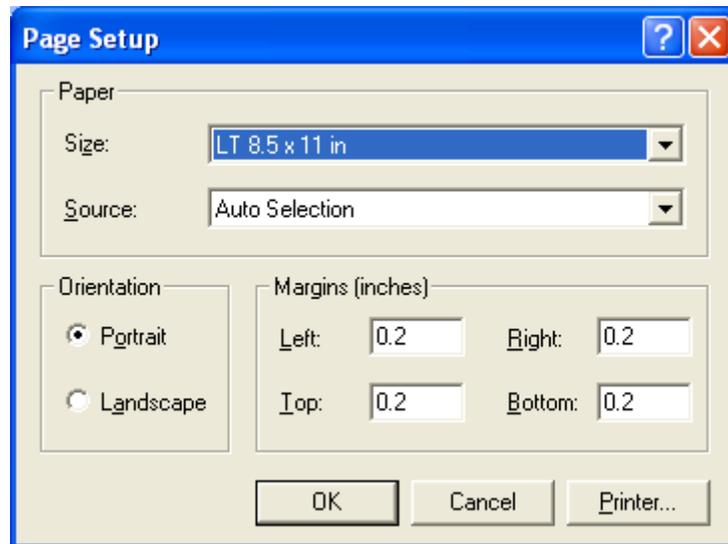
The menu option **File > Print** opens the standard print dialog for your operating system to send the currently viewed source file to the printer. The printout will use 66 lines per page.

By default, files are printed in the font and color shown on the screen when you choose the **File > Print** menu option.

You can customize code printing in the Options dialog. To open this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Advanced** category. See *Editor > Advanced Category* on page 107 for details about these fields.



To change the print output without changing the online display, choose the **File > Page Setup** from the menus. This dialog offers printing options similar to the following; they may differ depending on your operating system:



This chapter explains the architecture features provided by *Understand* and explains how you can use them to analyze your code.

This chapter contains the following sections:

Section	Page
About Architectures	176
Using the Architecture Browser	177
Viewing Architecture Dependency Graphs	179
Viewing Architecture Metrics	183
Managing Architectures	184
Creating an Architecture	185
Building an Architecture	187
Using XML to Manage Architectures	189

About Architectures

An architecture is an abstract hierarchy layered onto a body of source code. For example, a staff architecture could have nodes for each engineer working on a particular project. The nodes would contain a list of source code files belonging to or to be modified by that engineer. Dependencies and interactions could then be derived from that architecture.

Architectures allow you to name regions of a software project or ways of looking at software hierarchically. An architecture creates a hierarchy of source code units (entities). You can use the provided architectures or create your own.

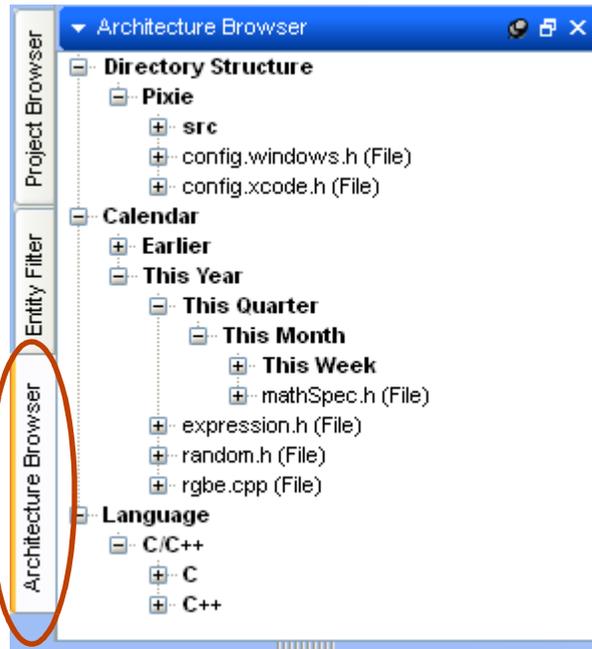
Architectures need not reference every source entity in the database; that is, they can define a subset of the entities. Also, architectures can contain a particular entity more than once. (Technically, that is, the architecture's flattened expansion need not maintain the set property.)

You can combine architectures successively to create novel filters for your entities.

From a more technical perspective, simple set algebra is used to combine and transform architecture hierarchies. The result of the filter is a list of entities. This result list can be viewed as a flat list or in terms of another architecture. The filter definition can be saved as a dynamic architecture. A dynamic filter architecture is updated as the contents of the database change and it can be used to reconstitute the filter at a later date.

Using the Architecture Browser

To open the Architecture Browser, choose **Project > Architectures > Browse Architectures** from the main menubar.



You see an expandable list of the architectures currently defined for your project.

This Architectures area is similar to the Filters area. When you click on an item, information about it is automatically shown in the Information Browser (as long as the “Sync” box is checked in the Information Browser).

Exploring Architectures

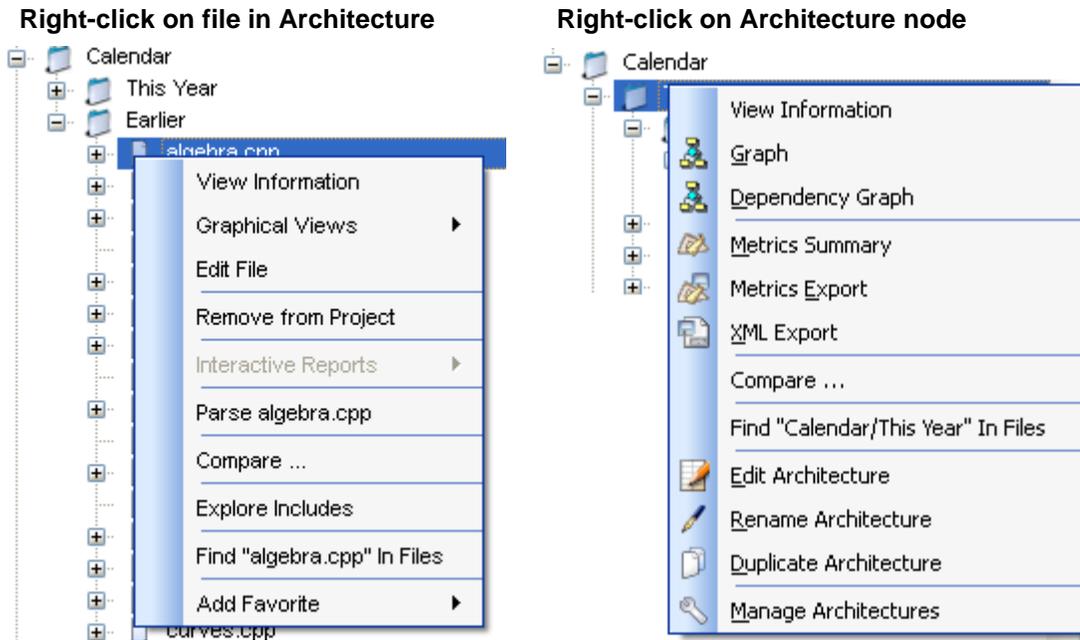
To explore the existing architectures, click the “+” signs to expand the hierarchy. Entities, such as files, functions, and variables are shown in the hierarchies.

Understand provides some “auto-architectures” that are built in:

- **Directory Structure:** Lists the project files in their normal file hierarchy—showing directories and their subdirectories.
- **Calendar:** Lists files in the project according to their last change date. A hierarchy of dates is shown that progresses from This Year, This Quarter, This Month, and This Week to Yesterday and Today.
- **Language:** Lists files first by their source code language and then by their location in the directory structure. (This architecture exists only if your project contains multiple languages.)

The auto-architectures are updated only when the project is analyzed. So, if your source code is actively being modified and you have not analyzed it recently, architectures—especially the Calendar architecture—could be out-of-date.

As always, you can right-click on any item in the Architecture Browser to get a list of information you can view about that item.



Notice that the context menu for an architecture node (such as a filesystem directory or “This Quarter” contains some extra items not available in other context menus:

- **Graphical Views > Graph Architecture:** Creates a graph of the architecture hierarchy from this point down. You are asked whether you want to include entities in the graph or just the architecture nodes. See page 179.
- **Graphical Views > Dependency Graphs:** Shows the dependencies between architecture nodes. See page 179.
- **Metrics Summary:** Provides metrics for the entities within the selected node. The metrics are based on entities in the current node, but not those in sub-nodes lower in the hierarchy. See page 183.
- **Metrics Export:** Creates a CSV output of the metrics from the Metrics Summary. See page 183.
- **XML Export:** Creates an XML export listing the architecture nodes and entities from the selected point down in the hierarchy. See page 189.
- **Edit Architecture:** Opens the Architecture Builder for the selected architecture if it is one you created. You cannot edit the auto-architectures provided with *Understand*. See page 187.
- **Rename Architecture:** Opens a Rename Architecture window that lets you rename the selected architecture or node if it is one you created. You cannot rename the auto-architectures provided with *Understand*. See page 185.
- **Duplicate Architecture:** Opens a Duplicate Architecture window that lets you type a name for a duplicate copy of the selected architecture. See page 185.
- **Manage Architectures:** Opens the Architect Manager window. See page 184.

Viewing Architecture Dependency Graphs

You can generate graphs that show the hierarchy of an architecture. You can save these graphs as PNG, JPEG, SVG, Visio XML, and DOT files.

Note: Dependency graphs are also available for classes and packages.

To create a graph, follow these steps:

- 1 Select the highest-level architecture node you want to graph. You can graph the entire hierarchy or just a sub-hierarchy.
- 2 Right-click on the node and choose **Graphical Views** from the context menu. Depending on the node you select, the submenu allows you to choose from **Graph Architecture**, **Depends On**, **Depended On By**, **Butterfly-Dependency Graph**, and **Internal Dependencies**. When you have selected an architecture node, the same list of graphical views is available by choosing **Graphs > Graphs for <current_entity>** from the menus.

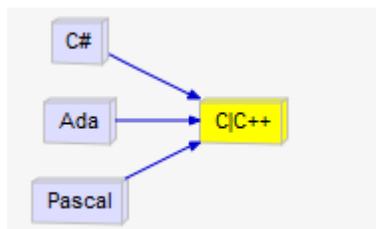
To open the Internal Dependencies graph for an entire architecture, choose from the **Graphs > Dependency Graphs** menu.

Architecture dependency graphs have the same toolbar as other types of graphical views. See page 233 for details about using the icons in the graphical view toolbar.



To save a graph as a JPG, PNG, or SVG file, see page 255. To save a graph to a Visio file, see page 256.

In all dependency graphs except the Graph Architecture view, there is a Graph Customizer pane that you can use to modify the graph display. This pane lets you control expansion, highlighting, and arrows on a per-node basis. It also lets you undo and redo your changes, and save and load graph customizations. For example, this is the default Depended On By graph for the C|C++ node in the multi-language zlib sample project.



Nodes that are drawn as 3D boxes (like those in the previous figure) can be expanded to show the nodes they contain by double-clicking on them. You can keep expanding nodes until you get to the file level.

You can right-click on a dependency graph to control whether long, short, or relative names are displayed for architecture node names and filenames. In addition, you can enable or disable the reference count numbers that show how many times a particular dependency occurs.

For a video that shows how to use the Graph Customizer, see <http://www.scitools.com/support/videos.php>.

Graph Customizer Toolbar

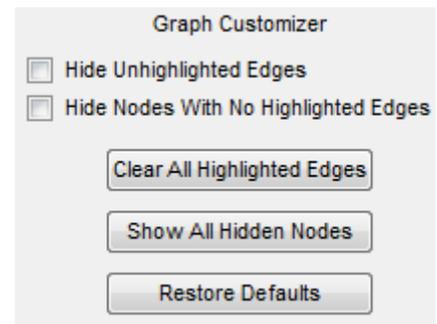
The toolbar icons in the Graph Customizer pane perform the following actions:



- **Save icon.** Prompts you for a name for the current settings. Settings that you can apply only to the specific graph type and root node in this view. If you have already saved settings for this graph type/root node combination, you can select a set you want to update from the drop-down menu. Otherwise, type a name for your current settings and click **Save**.
- **Load icon.** Prompts you to select a named group of graph settings that you want to open in the current window. The list shows only settings saved for this graph type/root node combination. To see the full list of saved settings, choose **Graphs > Dependency Graphs > Load Saved Dependency Graph**.
- **Undo icon.** Click this icon to undo your last change.
- **Redo icon.** Click this icon to redo the last change you undid.

Graph Customizer Fields

The first group of fields below the toolbar in the Graph Customizer pane performs actions that apply to all nodes in the view.



- **Hide Unhighlighted Edges.** This field is available only for Internal Dependency graphs, and you can use it only if you have turned on highlighting of “edges,” which are the connections between nodes. If you check this box, all arrows that are not highlighted are hidden, and the graph is reorganized as needed to omit those non-highlighted relationships.
- **Hide Nodes With No Highlighted Edges.** This field is available only for Internal Dependency graphs, and you can use it only if you have turned on highlighting of “edges.” If you check this box, all nodes that do not have a highlighted arrow pointing to it or away from it are hidden, and the graph is reorganized as needed to omit those nodes.
- **Clear All Highlighted Edges.** This button is available only for Internal Dependency graphs. If you click this button, all node and “edge” highlighting is removed.
- **Show All Hidden Nodes.** If you click this button, any nodes that have been hidden using the “Hide Selected Nodes” button are restored. This button does not expand any nodes that have been contracted to hide child nodes.
- **Restore Defaults.** If you click this button, the graph is restored to the settings it originally had when you opened it.

You can select one or more nodes in a dependency graph by using your mouse to drag a rectangle over the nodes you want to select. Or, hold down the Ctrl key while clicking on multiple nodes you want to select.

The fields in the Selected Node(s) area perform actions that apply only to the selected nodes.

- **Hide Selected Node(s).** Checking this box removes all the nodes that are currently selected from the graph and reorganized the graph as needed. (You can later restore the hidden nodes by clicking the Show All Hidden Nodes button.)
- **Show/Highlight Edges Going Out.** Checking this box for a node causes that node to be highlighted in yellow. Any arrows that point from this node to other nodes become darker, and nodes to which they point are highlighted in light blue. Internal Dependency graphs let you highlight such edges; other dependency graphs let you show or hide such edges.



Selected Node(s): C/C++

Hide Selected Node(s)

Selected Nodes Edges

Show Edges Going Out

Show Edges Coming In

Highlight Edges Going Out

Highlight Edges Coming In

Selected Nodes Children

Show Selected Node Children

Show Edges Between Children

Aggregate Child Edges Going Out

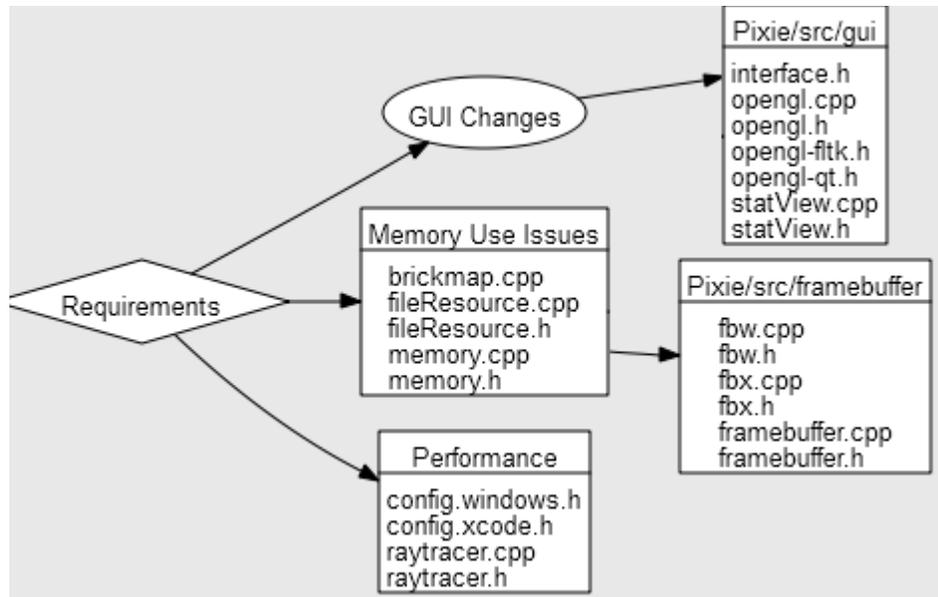
Aggregate Child Edges Coming In

- **Show/Highlight Edges Coming In.** Checking this box for a node causes that node to be highlighted in yellow. Any arrows that point to this node from other nodes become darker, and nodes which point to this node are highlighted in light blue. Internal Dependency graphs let you highlight such edges; other dependency graphs let you show or hide such edges.



- **Show Selected Node Children.** Checking this box for a node causes any child nodes to be displayed. This is the same as double-clicking on a node to expand it.
- **Show Edges Between Children.** Checking this box for a child node causes arrows to be drawn between this child node and any other child nodes as appropriate. If you remove the display of arrows, the graph is reorganized to hide these relationships.
- **Aggregate Child Edges Going Out.** If a node's children are shown and this box is checked, the arrows coming from the node's children are drawn as coming from the node, and arrows with the same target from multiple children are not repeated. Unchecking this box causes separate arrows to be drawn from the individual child nodes.
- **Aggregate Child Edges Coming In.** If a node's children are shown and this box is checked, the arrows going to the node's children are drawn as going to the node, and arrows to multiple children are not repeated. Unchecking this box causes separate arrows to be drawn too the individual child nodes.

The **Graph Architecture** view does not provide a Graph Customizer panel, but you can right-click on any dependency graph to modify the display. For example, in the following Architecture Graph, **Include Entity Lists** was off by default but was turned on by right-clicking.



Viewing Architecture Metrics

You can generate metrics information about an architecture or a subset of an architecture. The metrics information can be either a text summary or a comma-separated list for use in spreadsheets.

To create a metrics summary, follow these steps:

- 1 Select the highest-level node of the architecture for which you want metrics.
- 2 Right-click on the node and choose **Metrics Summary** from the context menu.
- 3 You see an Architecture Metrics Summary window. For example, the following two summaries use the Complexity architecture to compare metrics for “Low Complexity” and “High Complexity” functions.

Line	Complexity::Low - Architecture Metrics Summary	Complexity::High - Architecture Metrics Summary
1	CountLineCodeExe : 248.000000	66.000000
2		
3	AvgCyclomaticStrict : 1.315068	15.000000
4	CountStmtExe : 253.000000	59.000000
5	CountLine : 714.000000	116.000000
6	CountPath : 96.000000	1536.000000
7	CountLinePreprocessor : 3.000000	0.000000
8	CountSemicolon : 272.000000	57.000000
9	CountStmt : 292.000000	70.000000
10	AvgCyclomaticModified : 1.315068	14.000000
11	CountInput : 100.000000	6.000000
12	CountLineComment : 91.000000	19.000000
13	CountOutput : 132.000000	9.000000
14	MaxEssentialKnots : 0.000000	11.000000
15	AvgCyclomatic : 1.315068	14.000000
16	CountLineCodeDecl : 112.000000	12.000000
17	CountLineBlank : 100.000000	12.000000
18	CountLineInactive : 1.000000	0.000000
19	MaxNesting : 1.000000	2.000000
20	CountStmtDecl : 39.000000	11.000000
21	CountLineCode : 523.000000	90.000000
22		
23		

- 4 When you close the window, you are asked whether you want to save the file. If you click **Save**, you can save the summary as text.

To create a metrics export file, follow these steps:

- 1 Select the highest-level node of the architecture for which you want metrics.
- 2 Right-click on the node and choose **Metrics Export** from the context menu.
- 3 You see a comma-separated values file. The heading label for each column is in the first row. Each node in the architecture hierarchy has a separate row with metrics for that node's contents.

```

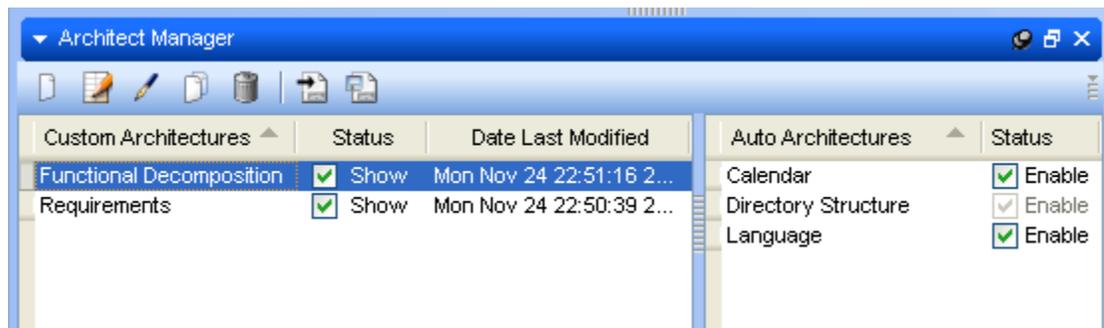
1 Name, CountLineCodeExe, AvgCyclomaticStrict, Cou...
2 Complexity, 593, 2.45055, 599, 1535, 2016, 9, 597, 2...
3 Complexity::Low, 248, 1.31507, 253, 714, 96, 3, 272, ...
4 Complexity::Medium, 279, 6.58824, 287, 705, 384, 6, ...
5 Complexity::High, 66, 15, 59, 116, 1536, 0, 57, 14, 70...
6

```

- 4 When you close the window, you are asked whether you want to save the file. If you click **Save**, you can save the data as a .CSV file.

Managing Architectures

To open the Architect Manager window, choose **Project > Architectures > Manage Architectures** from the main menubar in *Understand*. The window lists the auto-architectures on the right and custom architectures you have created on the left.



The checkboxes allow you to control whether custom and auto architectures are shown in the Architectures area. Removing the checkmark next to an architecture can improve performance, especially for large projects. So, you might want to disable/hide architectures you never or rarely use.

You can use the icons at the top of this area or right-click on an architecture to perform the following actions:

-  **Create a new architecture:** See page 185.
-  **Edit architecture:** Predefined and custom architectures only. See page 187.
-  **Rename architecture:** Predefined and custom architectures only. See page 185.
-  **Duplicate architecture:** See page 185.
-  **Delete architecture:** Predefined and custom architectures only.
-  **Import architecture from XML:** See page 189.
-  **Export architecture to XML:** See page 189.

Creating an Architecture

There are several ways to create a new architecture:

- To create an architecture from scratch, choose **Project > Architectures > New Architecture** from the menus or click the  icon in the Architect Manager. Use the Architecture Wizard to create the architecture as described in *Using the Architecture Wizard* on page 186.
- To duplicate an existing architecture (which you can then modify), select an architecture and click the  icon in the Architect Manager window. Or, right-click an existing architecture node in the Architecture Browser and choose **Duplicate Architecture** from the context menu to create an architecture from that node and lower in the hierarchy.

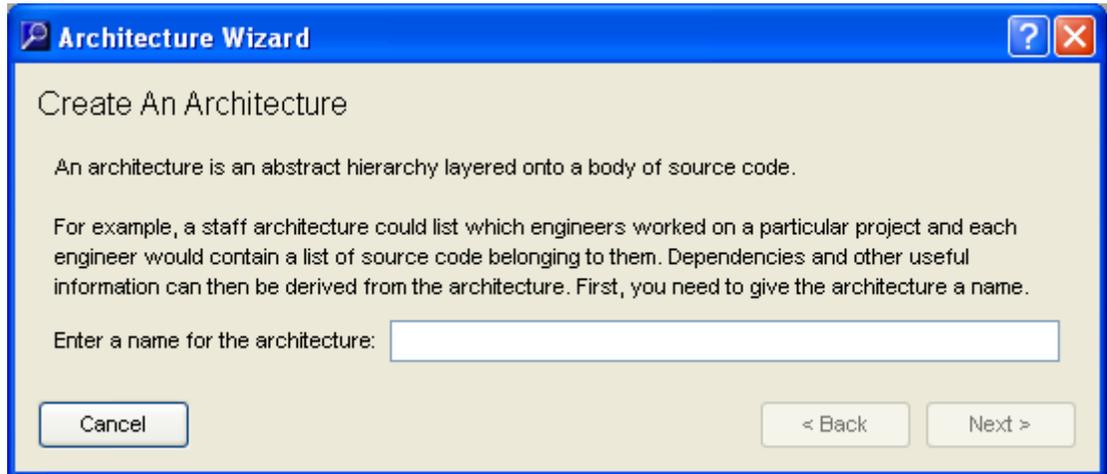


You can rename an architecture you have created by selecting an architecture and clicking the  icon in the Architect Manager window. Or, right-click on an existing custom architecture and choose **Rename Architecture** from the context menu.

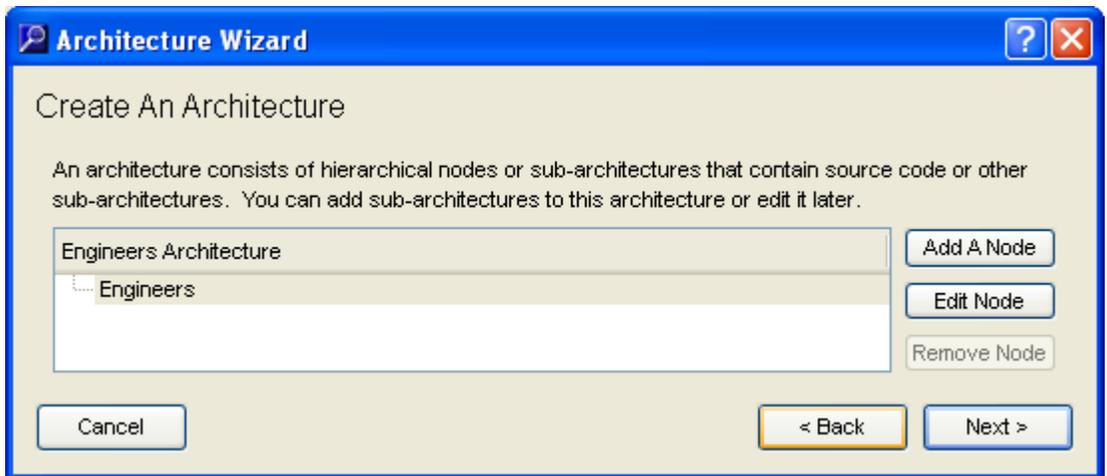
Using the Architecture Wizard

When you open the Architecture Wizard by choosing **Project > Architectures > New Architecture** from the menus or clicking the  icon in the Architect Manager window, you see a page that asks for the name of your architecture.

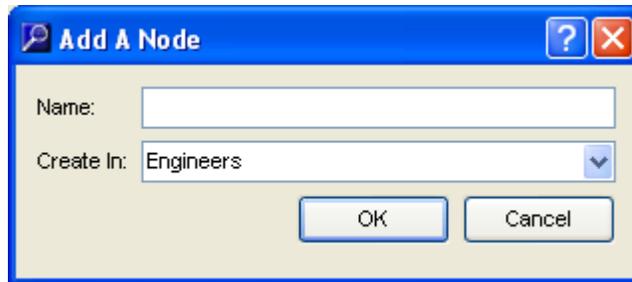
Type a name for the architecture. This name should be fairly short so it can be shown in architecture trees.



Then click **Next** to see the page that lets you add and edit architecture nodes. This is the hierarchy to which entities will be assigned in a later page of the wizard



Click **Add a Node** and type the **Name** of the node you want to add. The default location is within the node you had selected in the Architecture Wizard, but you can select another location in the **Create In** field. Then click **OK**.



You can modify nodes you have created by selecting a node and clicking **Edit Node**. You can delete the selected node by clicking **Remove Node**.

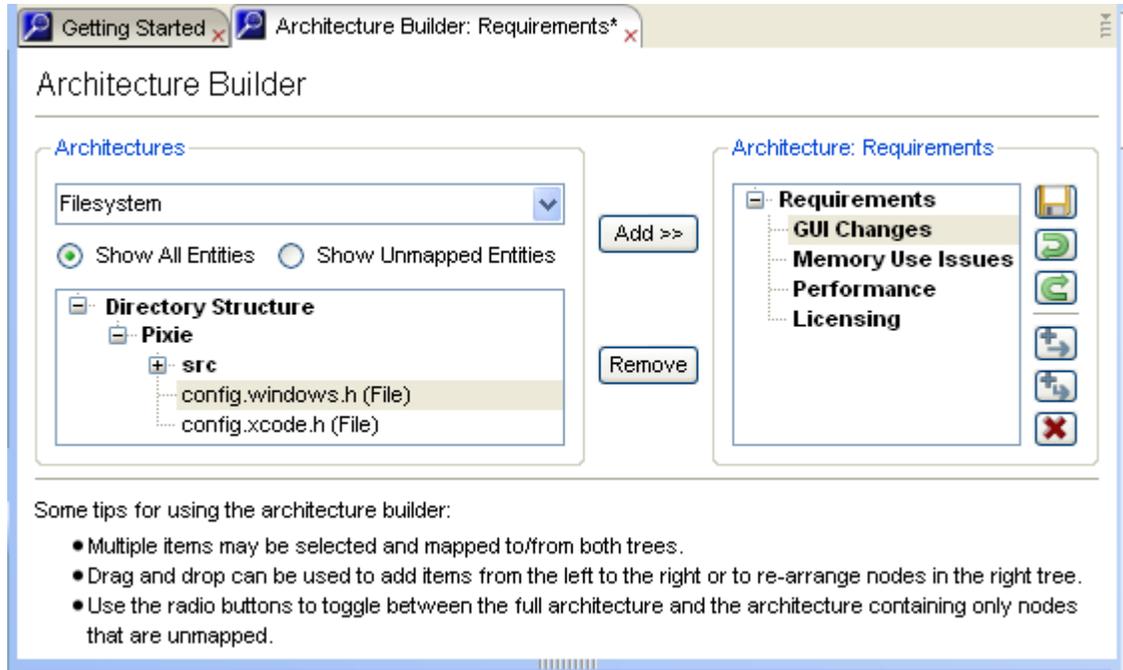
The next window presents an animation that shows how to use the Architecture Builder to add entities to the nodes you have created. When you have finished watching the animation, click **Finish**. This opens the Architecture Builder shown in the animation. Your architecture nodes are shown on the right. See *Building an Architecture* on page 187 for details on adding entities to each node.

Building an Architecture

To edit an existing custom architecture, select that architecture and click the  icon in the Architect Manager window. Or, right-click on an existing architecture and choose **Edit Architecture** from the context menu. Both actions open the Architecture Builder.

You cannot edit the Auto Architectures provided with *Understand*. However, you can use the  icon in the Architect Manager window to create a duplicate architecture of one of the Auto Architectures. Then, you can edit the duplicate architecture.

This dialog allows you to add nodes to architectures. You create an architecture structure on the right-hand side and map entities into the architecture from the left-hand side.



To create and edit nodes in the Architecture Builder, follow these steps:

- Double-click the name of any node on the right side of the Architecture Builder, and rename that node by typing. (Or you can select a node and press Enter.)
- Move one or more nodes by dragging them to the node you want to be the parent node. Within a node, the children are sorted alphabetically.
- Click the  icon to create a new node at the same level as the selected node. Click the  icon to create a new node as the child of the selected node.
- Click the  icon to delete the selected node.
- Click the  icon to undo your last change. Click the  icon to redo your last undo.

To map files to nodes in the Architecture Builder, follow these steps:

- 1 On the left side of the Architecture Builder, select an existing architecture from the drop-down list that will allow you to easily find the files you want. The default is the Directory Structure architecture.
- 2 You can choose whether to show all entries in the architecture or just the unmapped entries. For example, if you want to map all the entries into your new architecture, you might want to select **Show Unmapped Entries** so that you can see which files you haven't mapped yet.
- 3 In the left architecture hierarchy, select one or more files or architecture nodes.

- 4 In the right architecture hierarchy, select the node you want to contain your selection.
- 5 Click the **Add** button or drag your selection to the right side.
- 6 When you finish editing your custom architecture, click **Save**.

You can use the **Remove** button to delete files and nodes from the architecture you are editing.

As always, you can right-click on any node or file to use its context menu to get information.

You can save your edits to the architecture at any point by clicking the  icon. Then, you can continue editing. If you close the Architecture Builder without saving changes, you will be asked if you want to save your changes.

Using XML to Manage Architectures

You can use XML as a way to share architectures between one *Understand* database and another.

In addition to using XML to share architectures, you can use XML export/import to quickly create architectures that are a simple subset of another architecture by selecting a lower node in the hierarchy.

Exporting Architectures to XML

To create an XML file for an architecture, follow these steps:

- 1 Select the highest-level node of the architecture that you want to export. All of the hierarchy below the node you select will be represented in the XML file.
- 2 Click the  icon in the Architect Manager window. Or, right-click on the node you selected and choose **XML Export** from the context menu.
- 3 You see an XML file that contains <arch> and <set> tags for architecture nodes.
- 4 When you close the XML window, you are asked if you want to save the file. If you click **Save**, the default filename is the name of the node you selected.

Importing XML Architectures

To import an XML file for an architecture, follow these steps:

- 1 Click the  icon in the Architect Manager window.
- 2 In the Choose XML File to Import Architecture dialog, select an XML file that matches the tag format used by *Understand* to describe architectures. For example, you can choose XML files created by *Understand*. Click **Open**.
- 3 The architecture described by the XML file is added to your list of architectures.

This chapter describes how to create and view reports and the types of reports available.

This chapter contains the following sections:

Section	Page
Configuring Reports	191
Generating Reports	193
Viewing Reports	194
An Overview of Report Categories	195
Cross-Reference Reports	197
Structure Reports	202
Quality Reports	205
Metrics Reports	209

Configuring Reports

Understand provides a large number of reports you can generate about your code. These can be generated in HTML or text format. You can choose which reports and how to format them.

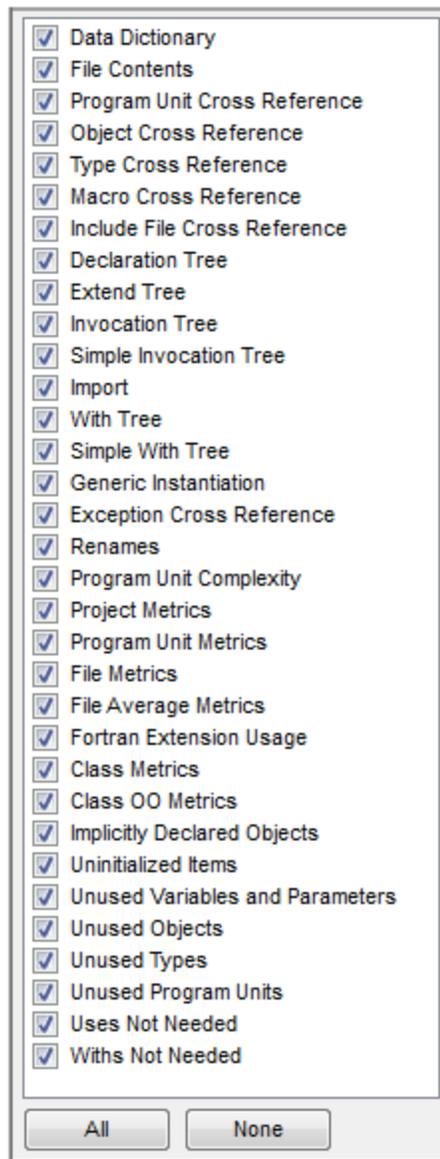
To configure how reports will be generated, choose **Reports > Configure Reports**. This opens the Project Configuration dialog with the **Reports > Output** category selected. From there, you can also configure the **Reports > Options** and **Reports > Selected** categories.

See page 54 for details on the **Reports > Output** category. In general, you can configure the following:

- **HTML reports:** The “home” file for the reports is `index.html`, but you can select an alternate title page. You may generate single or multiple HTML files for each report type. It is recommended that you split up the files for large projects. Choose *Alphabetic* to generate multiple HTML files per report that are split up alphabetically by the first letter of the entity name. Choose *Every n Entities* to generate multiple HTML files per report that are split up every “n” number of entities. By default, a single HTML file is generated for each letter of the alphabet.
- **Text reports:** You may generate one text file of the specified name (by choosing File). This one file will contain all the selected reports. Alternately, you may generate multiple text files (by choosing Separate Files) and specify a common filename prefix. The filenames of each text file identify the report.

For details on the **Reports > Options** category, see page 55.

The **Reports > Selected** category lets you select from the available reports for the languages used by your project. This list shows all the reports for all languages:



The specific reports available depend upon the source languages used in your project. See *An Overview of Report Categories* on page 195 for descriptions of the types of reports you can generate.

Customizing Report Colors

HTML reports use Cascading Style Sheets (CSS) to set colors and font styles used for keywords, comments, strings, numbers, and more. The colors and styles used are defined in the `sourcestyles.css` file, which is created the first time you generate HTML reports in a particular location.

You can customize the `sourcestyles.css` file using a text editor. Any colors and font styles normally supported by CSS can be used in this file. For example:

```
span.comment{color:DarkSeaGreen;font-style:italic}
```

If you modify the stylesheet and want to use it for other reports you generate, you can copy the modified `sourcestyles.css` file to the locations of other HTML reports.

Generating Reports

Once you have specified formatting options and the types of reports to be generated, choose **Reports > Generate Reports** from the menus to begin generating the selected reports. You see a dialog that shows the progress of the report generation.

On Windows, the ASCII text follows the DOS text file format (carriage return and line feed at the end of each line). On UNIX, text files are created according to the UNIX convention (lines end with a carriage return).

HTML reports are generated as HTML 3.0 format files. The generated HTML is not complex, the only HTML 3.0 (versus HTML 2.0) feature used is frames. Netscape 2.0 and higher, and Internet Explorer 3.0 and higher can display the files.

You can view the reports as described in *Viewing Reports* on page 194.

For large projects, reports can take a long time to generate. You can click **Cancel** to halt report generation. Clicking **Cancel** leaves the reports in a partially generated state.

Note: You may want to temporarily toggle off anti-virus protection programs while reports are being generated. This may speed the process of creating reports. If you do this, be sure to turn on virus checking after report generation is finished.

Note: HTML, text, and project metrics reports may also be generated with the “und” command line program. Refer to Chapter 14 for details.

Viewing Reports

To view generated reports, choose **Reports > View Reports**. Then choose the **HTML** or **Text** option.

File names of reports generated vary based on the type and format of the report generated.

- For text files, a single text file containing all selected reports may be generated or separate files for each type of report may be generated. A single text file is named `<project_name>.txt`. For separate text files, the file name is the type of report.
- For HTML reports, you can generate either a single HTML files for each report type, or smaller files divided either alphabetically by entity name or in groups of N number of entities. An index file is also generated that contains links to all the other HTML reports generated. The main window page is named `index.html`.

For HTML reports, a single index file contains an alphabetic list of all entities found in all other generated HTML reports. The entities listed in the index have hyperlinks to the Data Dictionary report for that entity. The entity index file is named `entity_index.html` and can be accessed from the “index” link on the main HTML page.

The following figure shows an example of the entity index.



An Overview of Report Categories

Understand generates a wide variety of reports. The reports fall into these categories:

- **Cross-Reference** reports show information similar to that in the *Information Browser*, except that all entities are shown together in alphabetic order. See *Cross-Reference Reports* on page 197.
- **Structure** reports show the structure of the analyzed program. See *Structure Reports* on page 202.
- **Quality** reports show areas where code might need to be examined. See *Quality Reports* on page 205.
- **Metrics** reports show basic metrics such as the number of lines of code and comments. See *Metrics Reports* on page 209.

The following table shows the type and page number for each report.

Report Type	Report Name and Page
Cross-Reference	<i>Data Dictionary Report</i> on page 197
Cross-Reference	<i>File Contents Report</i> on page 198
Cross-Reference	<i>Program Unit Cross-Reference Report</i> on page 198
Cross-Reference	<i>Object Cross-Reference Report</i> on page 199
Cross-Reference	<i>Type Cross-Reference Report</i> on page 199
Cross-Reference	<i>Macro Cross-Reference</i> on page 200
Cross-Reference	<i>Include File Cross-Reference</i> on page 200
Cross-Reference	<i>Exception Cross-Reference Report</i> on page 201
Structure	<i>Declaration Tree</i> on page 202
Structure	<i>Class Extend Tree</i> on page 203
Structure	<i>Invocation Tree Report</i> on page 203
Structure	<i>Simple Invocation Tree Report</i> on page 203
Structure	<i>Import Report</i> on page 204
Structure	<i>With Tree Report</i> on page 204
Structure	<i>Simple With Tree Report</i> on page 204
Structure	<i>Generic Instantiation Report</i> on page 204
Structure	<i>Renames Report</i> on page 204
Quality	<i>Program Unit Complexity Report</i> on page 205
Quality	<i>Uninitialized Items</i> on page 207
Quality	<i>Unused Variables and Parameters</i> on page 207
Quality	<i>Unused Objects Report</i> on page 207
Quality	<i>Unused Types Report</i> on page 207
Quality	<i>Unused Program Units Report</i> on page 208
Quality	<i>Uses Not Needed Report</i> on page 208
Quality	<i>Withs Not Needed Report</i> on page 208
Quality	<i>Implicitly Declared Objects Report</i> on page 207

Report Type	Report Name and Page
Quality	<i>FORTTRAN Extension Usage Report</i> on page 206
Metrics	<i>Project Metrics Report</i> on page 209
Metrics	<i>Program Unit Metrics Report</i> on page 210
Metrics	<i>File Metrics Report</i> on page 211
Metrics	<i>File Average Metrics Report</i> on page 212
Metrics	<i>Class Metrics Report</i> on page 210
Metrics	<i>Class OO Metrics Report</i> on page 210

Augment with the PERL or C API

The reports included with *Understand* have evolved over many years to accommodate common customer requests. However, we recognize that not all needs can be covered.

To help you develop custom reports we include both PERL and C interfaces to *Understand* databases.

For details on the PERL interface choose **Help > PERL API Documentation**. Also visit the blog and forum on our website.

The **Reports > Project Interactive Reports** and **Graphs > Project Graphs** commands display a list of user-created plugins, which can be created using the Perl API. For information about creating plugins, please contact support@scitools.com. The SciTools forum at <http://scitools.com/support/forum> and the SciTools blog at <http://scitools.com/blog> also contain messages concerning plugins.

Cross-Reference Reports

Cross-Reference reports show information similar to that in the References section of the Information Browser, except that all entities are shown together in alphabetic order. The following table shows the page that describes each type of cross-reference report.

Report Name
<i>Data Dictionary Report</i> on page 197
<i>Program Unit Cross-Reference Report</i> on page 198
<i>File Contents Report</i> on page 198
<i>Object Cross-Reference Report</i> on page 199
<i>Type Cross-Reference Report</i> on page 199
<i>Class and Interface Cross-Reference</i> on page 199
<i>Macro Cross-Reference</i> on page 200
<i>Include File Cross-Reference</i> on page 200
<i>Exception Cross-Reference Report</i> on page 201

Data Dictionary Report

The *Data Dictionary Report* lists all entities alphabetically. Each listing shows the entity name, what kind of entity it is (for example, macro, type, variable, function, include, file, or procedure), along with links to the location where each is declared in the source code.

The screenshot shows a navigation bar at the top with letters A through T. The letter 'E' is highlighted in purple. Below the bar, a list of entities is displayed:

- `echo` (Procedure) [\[xref\]](#)
[\[lexlib.pas, 284\]](#)
- `emp` (Variable) [\[xref\]](#)
[\[dept_empl.pas, 150\]](#)
- `employee` (Sql Table)
[\[dept_empl.pas, 9\]](#)
- `employee.age` (Sql Column)
[\[dept_empl.pas, 11\]](#)

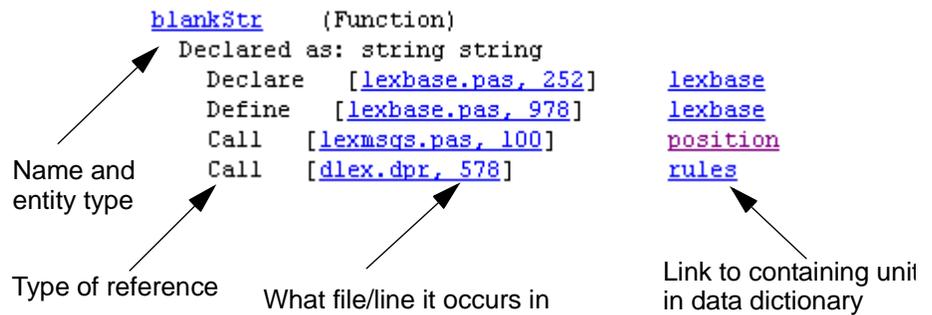
Arrows point from the following text to the corresponding parts of the screenshot:

- Name**: points to the entity name (e.g., `echo`).
- What kind of entity it is**: points to the entity type (e.g., (Procedure)).
- What file/line it was declared in.**: points to the file and line number link (e.g., [\[lexlib.pas, 284\]](#)).
- Quick link to cross-reference of this entity**: points to the [\[xref\]](#) link.
- Optionally break up report alphabetically**: points to the navigation bar.

Program Unit Cross-Reference Report

The *Program Unit Cross-Reference Report* lists all program units (such as procedures and functions) analyzed in alphabetic order along with information about what they return (if anything), what parameters are used, and where they are used by other program units.

The HTML version offers hyperlinks to the Data Dictionary report entry and to the source code where each reference occurs.



You can create an additional Program Unit Index report to list all the program units in the project and show the file and line where each is declared. For text reports, this is stored in a *.pcn file.

File Contents Report

Lists functions declared within a source file and the line numbers where they are declared. HTML versions of this report permit hyperlinked jumping to the function in the source as well as viewing of the entire source file.

```

regerror.c
Global Functions
regerror

regexp.c
Local Variables
regbol
regcode
regdummy
regendp
reginput
regmpar
regparse
regsize
regstartp

```

Object Cross-Reference Report

The Object Cross-Reference Report lists all objects (FORTRAN *variables*, *parameters*, *macros*) in alphabetic order along with declaration and usage references.

```

UMINUS      (Const)
  Declared as: 259
  Define   [expr.dpr, 19]      expr

undefined_symbol  (Const)
  Declared as: '202: undefined symbol'
  Define   [lexmsgs.pas, 67]      lexmsgs
  Use     [lexrules.pas, 179]    push\_macro
  Use     [lexrules.pas, 259]    add\_start\_state

```

The HTML version of this report includes hyperlinks to the Data Dictionary Report and the source code where the reference occurs.

Type Cross-Reference Report

The Type Cross-Reference Report lists all declared types in alphabetic order, along with their declaration and usage information. The HTML version of the report offers hyperlinks to the Types data dictionary report entry, as well as the source code where the reference occurs.

```

final      (Field)
  Declared as: Boolean
  Define   [lextable.pas, 99]      StateTableEntry

FirstPosTable  (Type)
  Declared as: array[0..2*max_start_states] of IntSetPtr
  Define   [lextable.pas, 89]      lextable
  Typed   [lextable.pas, 129]    first\_pos\_table

```

Class and Interface Cross-Reference

The Class and Interface Cross-Reference Report lists all declared classes and interfaces in alphabetic order, along with their declaration and usage information. The HTML version of the report includes hyperlinks to the data dictionary report entries, as well as the source code where the reference occurs.

```

Error      (Unknown Class)
  Declared as:
  Create   [REMatch.java, 62]      REMatch.clone
  Create   [Grep.java, 273]        Grep.processStream

Event      (Unknown Class)
  Declared as:
  Typed   [REApplet.java, 198]    e

Exception  (Unknown Class)
  Declared as:
  Extend   [REException.java, 38] regexp.REException
  Typed   [Grep.java, 236]        ex

```

Macro Cross-Reference

The Macro Cross-Reference Report lists all macros analyzed in the source code in alphabetic order along with information about where they are declared and where they are used. The HTML version offers hyperlinks to the macro's Data Dictionary report entry and to the source code where each reference occurs.

EXP

```
Declared as: CONST1
Define  [define_test.jov, 5]  define_test.jov
Use    [define_test.jov, 36]  SOMEPROC
```

Include File Cross-Reference

The Include File Cross-Reference Report lists all include files analyzed in the source code in alphabetic order with information about which files include them. The HTML version offers hyperlinks to the source code where each reference occurs.

Include File Cross Reference Report

Non-Alpha	A	B	<u>C</u>	D	E	F	G	H	I	J	K	L	M	N	O	P
-----------	---	---	----------	---	---	---	---	---	---	---	---	---	---	---	---	---

stdio.h

```
Include  [egrep.c, 45]      egrep.c
Include  [try.c, 32]       try.c
Include  [timer.c, 31]    timer.c
Include  [regsub.c, 21]   regsub.c
Include  [regexp.c, 25]   regexp.c
Include  [regerror.c, 1]  regerror.c
```

string.h

```
Include  [strpbrk.c, 1]    strpbrk.c
```

Exception Cross-Reference Report

The *Exception Cross-Reference Report* documents the declaration and usage of all exceptions. Each declaration and any *raises* or *handles* are documented. In the HTML version each raise or handle may be visited in the source, as well as the declaration point of the Exception (if visible).

EXCEPTION CROSS REFERENCE REPORT

queue_empty

Declared In: queue_package
FILE: [queue pa.adb LINE:21](#)
Referenced In:
RAISE FILE: [queue pa.adb LINE:113](#)
RAISE FILE: [queue pa.adb LINE:123](#)
RAISE FILE: [queue pa.adb LINE:133](#)

queue_full

Declared In: queue_package
FILE: [queue pa.adb LINE:22](#)
Referenced In:
RAISE FILE: [queue pa.adb LINE:98](#)
RAISE FILE: [queue pa.adb LINE:162](#)

stack_empty

Declared In: stack_package
FILE: [stack pa.adb LINE:21](#)
Referenced In:
RAISE FILE: [stack pa.adb LINE:114](#)
RAISE FILE: [stack pa.adb LINE:124](#)

Structure Reports

Structure reports are designed to help you understand the relationships between various entities. The following table shows the page in this chapter that describes each type of structure report.

Report Name and Page

Declaration Tree on page 202

Class Extend Tree on page 203

Invocation Tree Report on page 203

Simple Invocation Tree Report on page 203

With Tree Report on page 204

Simple With Tree Report on page 204

Generic Instantiation Report on page 204

Renames Report on page 204

Import Report on page 204

Declaration Tree

The *Declaration Tree* shows the declaration nesting of each program unit analyzed. Each nesting level is indicated by an indent with a vertical bar used to help align your eyes when viewing. Each nesting level is read as “declares”. In the HTML version of the report each program unit name is a hyperlink to its entry in the *Program Unit Cross-Reference Report*.

```
Package Body Occupants
| Procedure Put_View
| Procedure Look
| Procedure Get
| Function May_I_Get
| Procedure Drop
| Function May_I_Drop
| Procedure Inventory
| Procedure Go
| | Block
```

In the above example, Package Body *Occupants* is the top level program unit. It has declared within it, *Put_View*, *Look*, *Get*, *May_I_Get*, *Drop*, *May_I_Drop*, *Inventory*, and *Go*. Nested within *Go* is an unnamed declare block.

The Declaration Tree report shows a textual representation of an declaration tree for each FORTRAN file.

Declaration Tree Report

```

allocate.f (File)
| allocate (Subroutine)
| | get01 (Subroutine)
| | daopen (Subroutine)
| | | lenchr (Subroutine)
| | | get012 (Subroutine)
| | | daopn2 (Subroutine)
| | | | prmpfl (Subroutine)
| | | | getyno (Subroutine)
| | | opntdd (Subroutine)
| | | opnrdr (Subroutine)
| | | dsrctl (Subroutine)

```

Class Extend Tree

The *Class Extend Tree* report shows the nesting of class declarations in the files analyzed. Each nesting level is indicated by an indent with a vertical bar to help align your eyes when viewing. Each nesting level is read as “extends”. In the HTML version of the report each class name is a hyperlink to its entry in the *Data Dictionary and Interface Cross-Reference Report*.

Invocation Tree Report

The Invocation Tree Report shows a textual representation of the invocation tree for each program unit analyzed. The report shows who each program unit calls. Levels are indicated by tabs and are lined up with vertical bars. Each nesting level is read as “calls”.

The HTML version offers hyperlinks to the corresponding Data Dictionary report entries.

```

definedKey
| hash
| | length
| | Ord
| | Inc
| exit
| newPos
| | Inc

```

Simple Invocation Tree Report

The Simple Invocation Tree Report shows the invocation tree to only one level for each program unit that has been analyzed.

The invocation level is indicated by an indent and a vertical bar and is read as “calls”.

```

definedKey
| hash
| exit
| newPos

```

With Tree Report

Structured identically to the other hierarchy reports, the *With Tree* report shows a textual version of the With Tree for each program unit that is not Withed by another.

As with the other textual hierarchy reports, indents show level with a vertical bar helping align your eye. For this report, each line is read as “Withs”.

```

Package Body Occupants
| Package Rename Text_IO
| | Package Text_IO
| | | Package IO_Exceptions
| | | Package System
| | | Package Parameters

```

In the above example, the package body *Occupants* Withs package *Text_IO*, which in turn Withs *IO_Exceptions*, *System*, and *Parameters*.

Simple With Tree Report

The *Simple With Tree* report is similar to the *With Tree* report. It shows a textual representation of the With Tree for each program unit that is not Withed by another. However, it shows only one level of withs. For example:

```

Package Body Occupants
| Package Rename Text_IO

```

Generic Instantiation Report

This report lists each package that was created through instantiation.

In the HTML version, the source where it was instantiated and its Data Dictionary Report entry may be visited from hyperlinks.

```

My_Int_IO Package Instantiation
          FILE: board.adb LINE:12
          Instantiated From => INTEGER_IO Generic Package

```

Renames Report

The *Renames Report* cross-references the use of the Ada command “renames”, as in:

```
function Rouge return Color renames Red;
```

This report lists program units that have been renamed in alphabetic order. Each rename shows the program unit it renames, and in the HTML report a hyperlink to the rename instance in the source is provided.

The Information Browser also identifies packages and program units that rename others or are renamed.

Import Report

The *Import* report lists all source files that import other files and the files they import. The HTML version offers hyperlinks to the data dictionary entry for each imported file.

Quality Reports

Understand's quality reports are designed to provide information about areas of the analyzed source that might not meet standards or that hold the potential for trouble. They also identify areas where extra programming has been done but not needed. This sometimes identifies areas that aren't yet complete, or that haven't been maintained completely.

The following table shows the page in this chapter that describes each type of quality report.

Report Name and Page

<i>Program Unit Complexity Report</i>	on page 205
<i>FORTTRAN Extension Usage Report</i>	on page 206
<i>Implicitly Declared Objects Report</i>	on page 207
<i>Uninitialized Items</i>	on page 207
<i>Unused Variables and Parameters</i>	on page 207
<i>Unused Objects Report</i>	on page 207
<i>Unused Types Report</i>	on page 207
<i>Unused Program Units Report</i>	on page 208
<i>Uses Not Needed Report</i>	on page 208
<i>Withs Not Needed Report</i>	on page 208

The complete list of quality metrics available in *Understand* changes frequently - more frequently than this manual is reprinted. A complete and accurate list is always available on our web site: <http://www.scitools.com/documents/metrics.php>.

Program Unit Complexity Report

The *Program Unit Complexity Report* lists every procedure and function or similar program unit in alphabetic order along with the McCabe (Cyclomatic) complexity value for the code implementing that program unit.

The Cyclomatic complexity is the number of independent paths through a module. The higher this metric the more likely a program unit is to be difficult to test and maintain without error.

The Modified column shows the cyclomatic complexity except that each case statement is not counted; the entire switch counts as 1.

The Strict column shows the cyclomatic complexity except && and || also count as 1.

The Nesting column shows the maximum nesting level of control constructs in this program unit.

Program Unit Complexity Report

Click column header for explanation of each metric

	Cyclomatic	Modified	Strict	Nesting
Act	1	1	1	0
act_char	3	3	3	2
act_char	7	7	7	3
add_action	2	2	2	1
add_item	2	2	2	1
add_lit	3	3	3	1

FORTRAN Extension Usage Report

This report lists anywhere your source code has non-standard FORTRAN extensions. The report factors in what variant (F77, F90, F95) you chose on your project configuration.

The following is a snippet from a sample FORTRAN Extension Usage report:

Fortran Extension Usage Report

```

AUTOMATIC:          0
CEXTERNAL:          0
CLOSE Statement No-Parens: 0
DATAPOOL:           0
Declaration /clist/ initialization: 0
EXIT DO Statement:  0
EXIT IF Statement:  0
EXIT FOR Statement: 0
EXIT LOOP Statement: 0
EXIT WHILE Statement: 0
FOR Statement:      0
IF block without THEN: 0
IMPLICIT UNDEFINED: 0
LOOP Statement:     0

```

Implicitly Declared Objects Report

The *Implicitly Declared Objects Report* lists any variables or parameters that were implicitly declared using FORTRAN's implicit declaration mode. Using implicitly declared variables is considered a risky practice, and this report helps you weed out where the practice is occurring in your code.

The HTML version offers hyperlinks to the function's Data Dictionary report entry.

Uninitialized Items

The *Uninitialized Items* report lists items such as variables that are not initialized in the code. The report is organized by file. Each uninitialized item within the file is listed by name along with the line number on which the item is declared. The HTML version offers hyperlinks to the location where the item is declared.

Unused Variables and Parameters

The *Unused Variables and Parameters* report lists items that are declared (and perhaps initialized) but never referenced other than that. The report is organized by file. Each unused item is listed by name along with the type of item and the line number on which the item is declared. The function or similar container is shown after the list of unused items within it. Types of items may include functions, parameters, variables, and objects. The HTML version offers hyperlinks to the location where each unused item is declared.

Unused Objects Report

The *Unused Objects Report* lists objects (for example, variables, parameters, constants) that are declared but never used. The HTML version has links to the function's Data Dictionary report entry and to the source line where the object is declared.

dyacc.dpr		
	yymaxtoken	518
expr.dpr		
	UMINUS	19
	yymaxtoken	129
lexmsgs.pas		
	invalid charnum	68
	mem overflow	75

Unused Types Report

The *Unused Types Report* lists types that are declared but never used. The HTML version has links to the function's Data Dictionary report entry and the source where the type is declared.

dlib.pas		
	TLexerParserBase	33
exprlex.pas		
	cc	42

Unused Program Units Report

The *Unused Program Units Report* identifies program units that are declared but never used.

Note that this listing in this report doesn't mean the system doesn't need this program unit. For instance, interrupt handlers that are called by system interrupts are often never "used" within the other source of the program.

dlib.pas		
	parse	35
dyacc.dpr		
	yycharsyn	2017
expr.dpr		
	yycharsyn	468

Uses Not Needed Report

The *Uses Not Needed Report* identifies any unneeded "use" statements that provide access to a module's public specifications and definitions. To remove unneeded access, you may add only clauses to use statements.

Withs Not Needed Report

This report lists, any With statements a program unit has but does not need (by not using items made public by the With statement).

Note that this covers only direct usage in the program unit and doesn't account for side effects that may be needed by the program to operate correctly. For instance, sometimes a package can be Withed just to start a task or to execute code in its begin/end block.

Metrics Reports

Metrics provide statistical information about your project and entities, such as the number of lines of code and the complexity of various entities.

Understand provides a number of ways to gather metrics information. This section describes reports that provide metrics. See page 214 for other ways to gather metrics.

The following table shows the page in this chapter that describes each type of metrics report.

Report Name and Page

Project Metrics Report on page 209

Class Metrics Report on page 210

Class OO Metrics Report on page 210

Program Unit Metrics Report on page 210

File Metrics Report on page 211

File Average Metrics Report on page 212

The complete list of metrics available in *Understand* changes frequently—more frequently than this manual is reprinted.

A complete and accurate list is always available on our web site:

<http://www.scitools.com/documents/metrics.php>.

Project Metrics Report

The *Project Metrics Report* provides metric information about the entire project. The metrics reported include: the total number of files, the total number of program units, and the total number of lines of source code.

Project Metrics Report

Files:	23
Subroutines:	314
Lines:	12097
Blank Lines:	1351
Code Lines:	8695
Comment Lines:	3607
Declarative Statements:	1190
Executable Statements:	3373
Ratio Comment/Code:	0.41

These metrics are also reported on the title page of the HTML report.

Class Metrics Report

The *Class Metrics Report* provides the following metrics for each class that has been analyzed:

- Total number of lines
- Total number of blank lines
- Total number of lines of code
- Total number of lines that contain comments
- Average number of lines per class
- Average number of comment lines per class
- Average complexity per class
- Maximum complexity within class
- Ratio of comment lines to code lines

Class	Lines	Lines Blank	Lines Code	Lines Comment	Average Lines	Average Lines Comment	Average Complexity	Maximum Complexity	Ratio Comment/Code
TLexer	4	0	4	0	0	0	0	0	0.00
TLexer	305	0	297	18	301	18	9	9	0.06

Class OO Metrics Report

The *Class OO Metrics Report* provides the following object-oriented metrics for each class that has been analyzed:

- **LCOM (Percent Lack of Cohesion):** 100% minus the average cohesion for class data members. A method is cohesive when it performs a single task.
- **DIT (Max Inheritance Tree):** Maximum depth of the class in the inheritance tree.
- **IFANIN (Count of Base Classes):** Number of immediate base classes.
- **CBO (Count of Coupled Classes):** Number of other classes coupled to this class.
- **NOC (Count of Derived Classes):** Number of immediate subclasses this class has.
- **RFC (Count of All Methods):** Number of methods this class has, including inherited methods.
- **NIM (Count of Instance Methods):** Number of instance methods this class has.
- **NIV (Count of Instance Variables):** Number of instance variables this class has.
- **WMC (Count of Methods):** Number of local methods this class has.

Program Unit Metrics Report

The *Program Unit Metrics Report* provides information on various metrics for each program unit that has been analyzed.

	Lines	Blank Lines	Code Lines	Lines- exe	Lines- dec	Comment Lines	Ratio Comment/Code
act_char	10	0	10	7	3	1	0.10
act_char	84	1	78	60	16	10	0.13

The following metrics are provided for each program unit:

- **Lines:** Total number of lines in the function.
- **Comment:** Number of comment lines in the function.
- **Blank:** Number of blank lines in the function.
- **Code:** Number of lines in the function that contain any code.
- **Lines-exe:** Lines of code in the function that contain no declaration.
- **Lines-decl:** Lines of code in the function that contain a declaration or part of a declaration.
- **Stmt-exe:** Number of executable statements in the function.
- **Stmt-decl:** Number of declarative statements in the function. This includes statements that declare classes, structs, unions, typedefs, and enums.
- **Ratio Comment/Code:** Ratio of comment lines to code lines.
(comment_lines/code_lines)

Note: code+comment+blank != lines
Some lines may contain both code and comments.

File Metrics Report

The *File Metrics Report* provides information similar to that in the *Program Unit Metrics Report*. However, it is organized by file rather than by *program unit*.

Click on each metric column to get a detailed description of it.

Note: code+comment+blank != lines
Some lines may contain both code and comments.

	Lines	Blank Lines	Code Lines	Lines-exe	Lines-dec	Comment Lines	Execution Statements
dlex.dpr	797	86	636	498	112	104	387
dlib.pas	42	13	9	0	9	20	0

Declaration Statements	Mains / Subroutines	Avg. Complexity	Avg. Complexity Case 1	Ratio Comment/Code
46	21	6	6	0.16
3	0	0	0	2.22

.....
File Average Metrics Report

The *File Average Metrics Report* provides averages for the functions within a file. All lines outside any function are ignored when calculating the averages. The following metrics are provided for each function:

- **Cyclomatic:** The average number of independent paths through the functions in this file. The higher this metric the more likely a program unit is to be difficult to test and maintain without error.
- **Modified:** Same as Cyclomatic complexity except that each case statement is not counted; the entire switch statement counts as 1.
- **Strict:** Same as Cyclomatic complexity except that && and || also count as 1.
- **Essential:** Measures the amount of unstructured code in a function.
- **Lines:** Average number of lines in the functions in this file.
- **Code:** Average number of lines that contain any code in the functions in this file.
- **Comment:** Average number of comment lines in the functions in this file.
- **Blank:** Average number of blank lines in the functions in this file.

This chapter describes how to create and view metrics and the types of metrics available.

This chapter contains the following sections:

Section	Page
About Metrics	214
Metrics Summary	215
Metrics Browser	216
Exporting Metrics to HTML	217
Exporting Metrics to a CSV File	218
Configuring Metric Charts	220
Using the Metrics Treemap	223
Exporting Dependency Metrics	225

About Metrics

Understand provides a number of ways to gather metrics information:

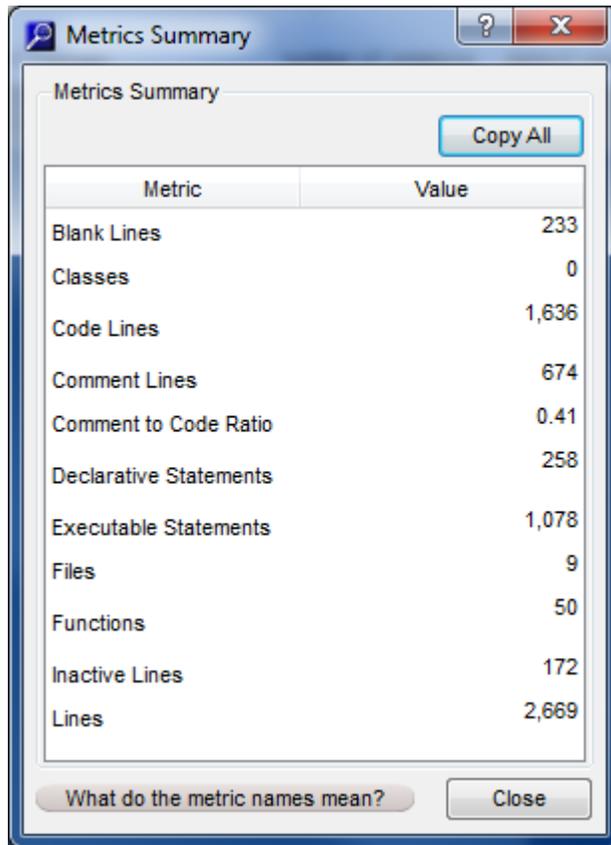
- **Information Browser:** The Information Browser tree has a **Metrics** node. You can expand this branch to show a few metrics for the current entity. See page 129.
- **Metrics Summary:** You can choose **Metrics > Metrics Summary** from the menus to see a short list of metrics for the entire project. See page 215.
- **Metrics Browser:** You can choose **Metrics > Browse Metrics** from the menus to see a browser that lets you choose any architecture node, file, or entity to see all the metrics available for that item. See page 216.
- **Export to HTML:** You can click this button in the Project Metrics Browser to export the full list of metrics for all architecture nodes and files. See page 217.
- **Export to CSV:** You can choose **Metrics > Export Metrics** from the menus to create a text file of all the project metrics in comma-delimited format. See page 218. (You can schedule this export to occur regularly; see page 50.)
- **Configure Metric Charts:** You can choose **Metrics > Configure Metric Charts** from the menus to open a dialog that lets you display graphs of volume and complexity metrics on an architecture basis. See page 220.
- **Reports:** When you create reports by choosing **Project > Project Reports**, some of the reports provide metrics. See page 209.
- **PERL/C API:** A more advanced way to get existing metrics and calculate new metrics is with the PERL and C API. These provide full access to the *Understand* database. Choose **Help > PERL API Documentation** for more information. See page 196.

Understand provides a large number of metrics you can generate about your code. The complete list of metrics available in *Understand* changes frequently—more frequently than this manual is reprinted.

A complete and accurate list of metrics is always available on our web site at <http://www.scitools.com/documents/metrics.php>. The “What do the metric names mean?” buttons in metrics dialogs links to this page.

Metrics Summary

Choose **Metrics > Metrics Summary** from the menus to see a short list of metrics for the entire project.



The screenshot shows a dialog box titled "Metrics Summary" with a "Copy All" button at the top right. Below the button is a table with two columns: "Metric" and "Value". The table lists various metrics and their corresponding values. At the bottom of the dialog, there are two buttons: "What do the metric names mean?" and "Close".

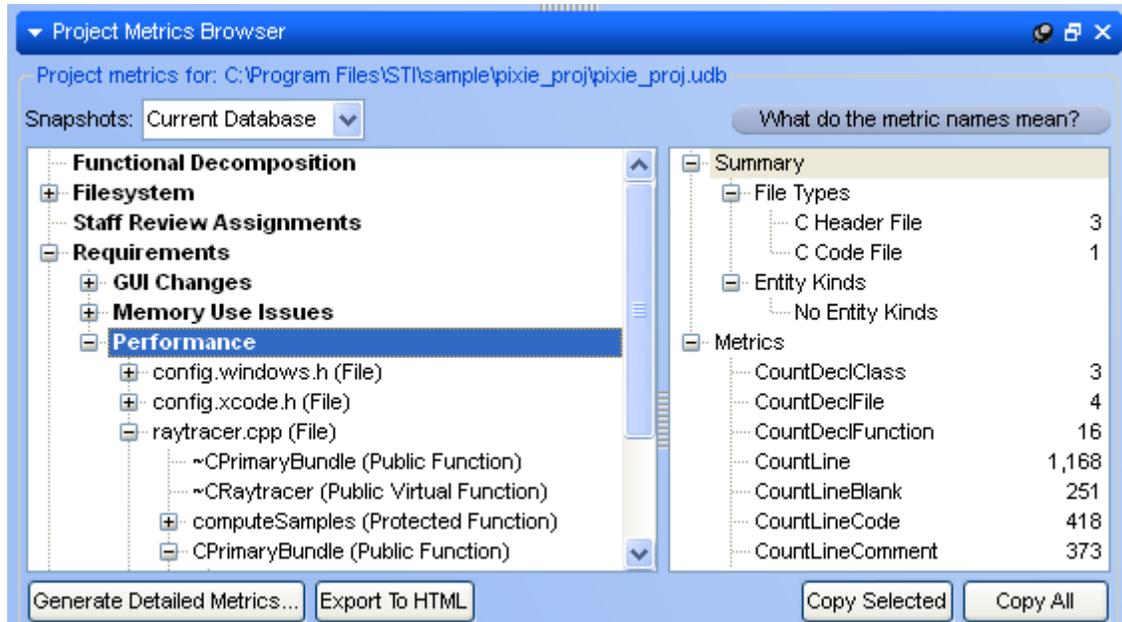
Metric	Value
Blank Lines	233
Classes	0
Code Lines	1,636
Comment Lines	674
Comment to Code Ratio	0.41
Declarative Statements	258
Executable Statements	1,078
Files	9
Functions	50
Inactive Lines	172
Lines	2,669

A complete and accurate list of metrics is always available on our web site at <http://www.scitools.com/documents/metrics.php>. The "What do the metric names mean?" buttons in metrics dialogs link to this page.

Metrics Browser

To open the Project Metrics Browser, choose **Metrics > Browse Metrics** from the menus.

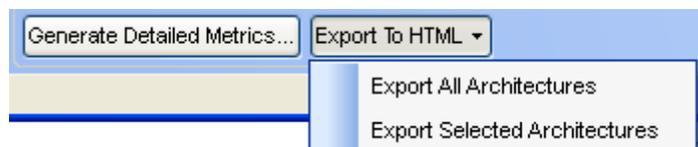
In this window, you can browse the architectures in your project and select any architecture node, file, or entity. The list on the right shows code size and complexity metrics for the selected item.



Double-click a file or entity to open the Source Editor for that item. Right-click to see the standard informational menu choices.

You can select rows on the right and click **Copy Selected** or press Ctrl+C to copy those lines to the clipboard. Click **Copy All** to copy the full list of metrics for the selected directory or file.

Click **Export to HTML** to generate reports as described on page 217. You can choose to **Export All Architectures** and all of their nodes or to **Export Selected Architectures** and nodes that you have selected using the Ctrl or Shift key while clicking on the nodes you want in the metrics export. Click **Generate Detailed Metrics** to open the Export Metrics dialog and generate a text file in comma-delimited format as described on page 218.



Exporting Metrics to HTML

You can click the **Export to HTML** button in the Project Metrics Browser (page 216) to export the full list of metrics for all entities and architecture nodes. When you click this button, you see a Browse for Folder dialog appropriate to your operating system.

Choose or create the folder where you want the metrics files to be created. The files are actually stored in a folder called “pixie_proj_Metrics” below the folder you select.

If the directory already exists, you are asked if the files should be overwritten. If you answer “No”, a number is appended to the old directory name to it to save it as a backup.

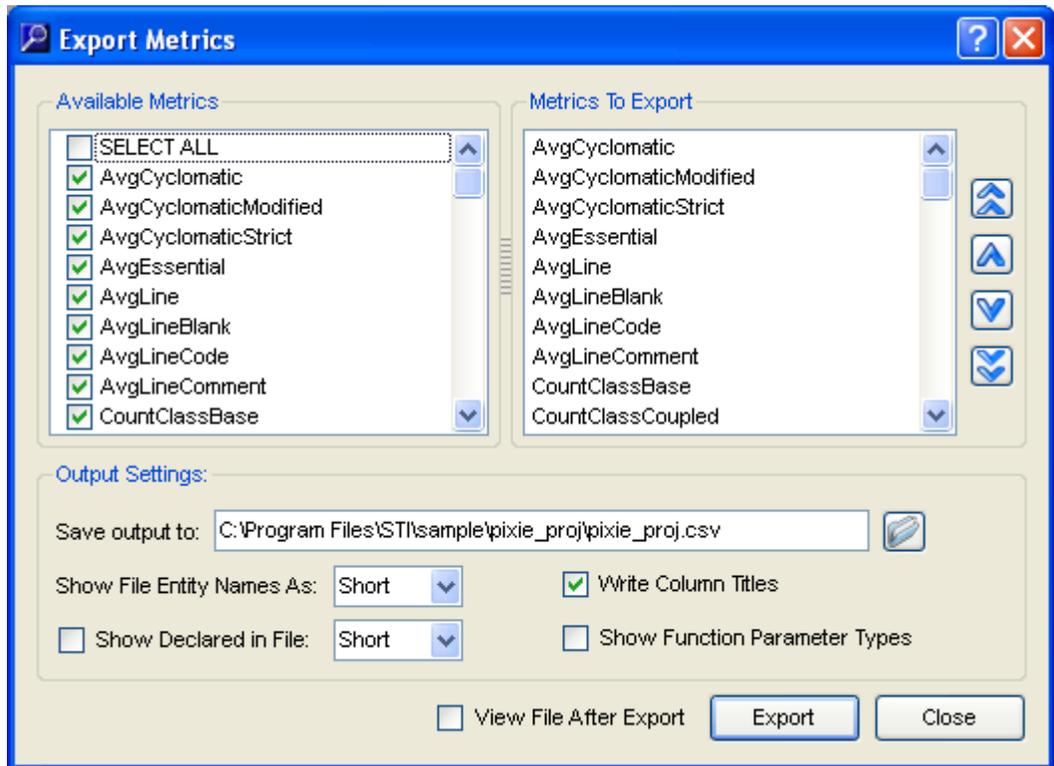
If the report is generated successfully, you are asked if you want to view the report. Click **Yes** to open the top-level page, index.html.

Architecture: Filesystem		File: mathSpec.cpp	
Summary		Metrics	
File Types			
C Header File	114	MaxCyclomaticModified	3
C Code File	100	CountLineCodeExe	148
Entity Kinds		AvgCyclomaticStrict	1
<i>None</i>		SumCyclomaticStrict	15
Metrics		CountStmtExe	160
CountLineComment	22,583	SumCyclomatic	15
CountLineInactive	14,813	AvgLineComment	0
RatioCommentToCode	0.05	AvgLine	18
CountDeclFile	214	CountLine	319
CountLineBlank	18,505	CountDeclClass	0
CountStmtDecl	15,588	RatioCommentToCode	0.42
CountDeclFunction	2,139	CountLinePreprocessor	8
CountLine	475,512	CountSemicolon	172
CountStmtExe	38,370	SumCyclomaticModified	15
CountLineCode	413,228	SumEssential	12
CountDeclClass	244	AvgCyclomaticModified	1
		CountStmt	187

The HTML-based report lets you select any architecture node, file, function, or other entity type that has metrics in the left pane. The right pane shows metrics available for that item.

Exporting Metrics to a CSV File

You can save metric information to a comma-delimited text file by choosing **Metrics > Export Metrics** from the menus or clicking the **Generate Detailed Metrics** button in the Project Metrics Browser. You can use the generated file in Excel and other spreadsheet programs. The Export Metrics dialog looks like this:



The defaults in this dialog come from the Project Configuration dialog in the **Metrics > Options** category (page 52) and the **Metrics > Selected** category (page 53).

You can override the defaults using the following fields:

- **Available Metrics:** Check the boxes next to metrics you want to include in the output. Check the “SELECT ALL” box to select all metrics. Uncheck the “SELECT ALL” box to unselect all metrics.
- **Metrics to Export:** Click the single arrow to move the selected metric up or down one in the list. Click the double arrow to move the selected metric to the top or bottom of the list.
- **Save output to:** Specify the location and name of the file you want to use for metrics output. *Understand* sends its metrics output to a .csv (comma-separated values) file.
- **Show File Entity Name as:** Specify whether files should be displayed with **Short** names (just the filename), **Full** names (including the absolute path), or **Relative** names (relative directory path).

- **Show Declared in File:** Check this box if you want the file in which each entity is declared to be included in the output. You can specify whether you want these files displayed with **Short** names, **Full** names, or **Relative** names.
- **Write Column Titles:** Check this box if you want column headings in the CSV file.
- **Show Function Parameter Types:** Check this box if you want the type of each function parameter listed.

After setting options, click **Export** to export the .CSV file. If you check the **View File After Export** box before exporting the file, the CSV file is opened with the default application for working with CSV files. This is likely to be a spreadsheet application.

If the output file already exists, you are asked if the files should be overwritten. If you answer “No”, you can change the output filename and click **Export** again.

You can schedule this metrics to be automatically exported to a CSV file on a regular basis. See page 50 for details.

See *Exporting Dependency Metrics* on page 225 for more types of metrics you can export to a CSV file.

A complete and accurate list of the available metrics is available at:

<http://www.scitools.com/documents/metrics.php>.

Configuring Metric Charts

Commands in the Metrics menu provide fast access to metrics charts for the current version of the entire project. These commands are:

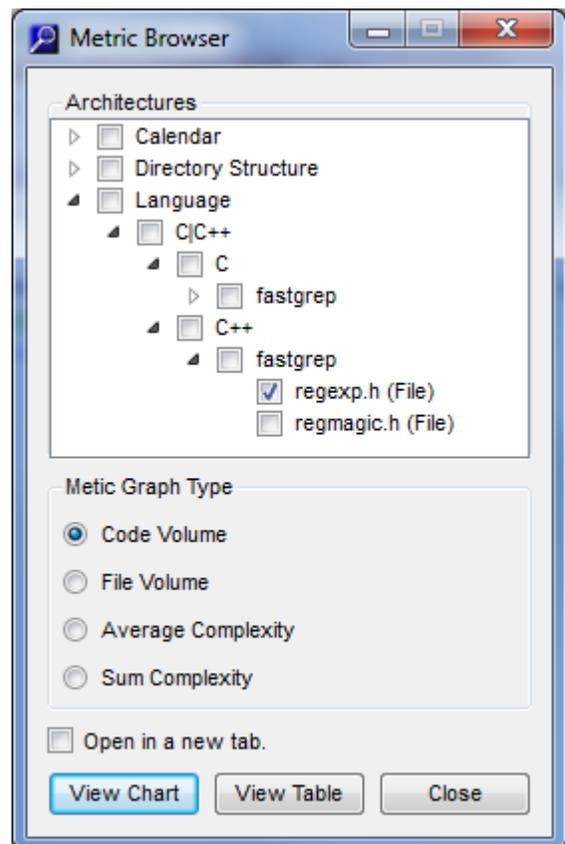
- **Metrics > Project Metric Charts > Code Volume**
- **Metrics > Project Metric Charts > File Volume**
- **Metrics > Project Metric Charts > Average Complexity**
- **Metrics > Project Metric Charts > Sum Complexity**

You can choose **Metrics > Configure Metric Charts** from the menus to open the Metric Browser, which lets you display graphs of various metrics on an architecture basis.

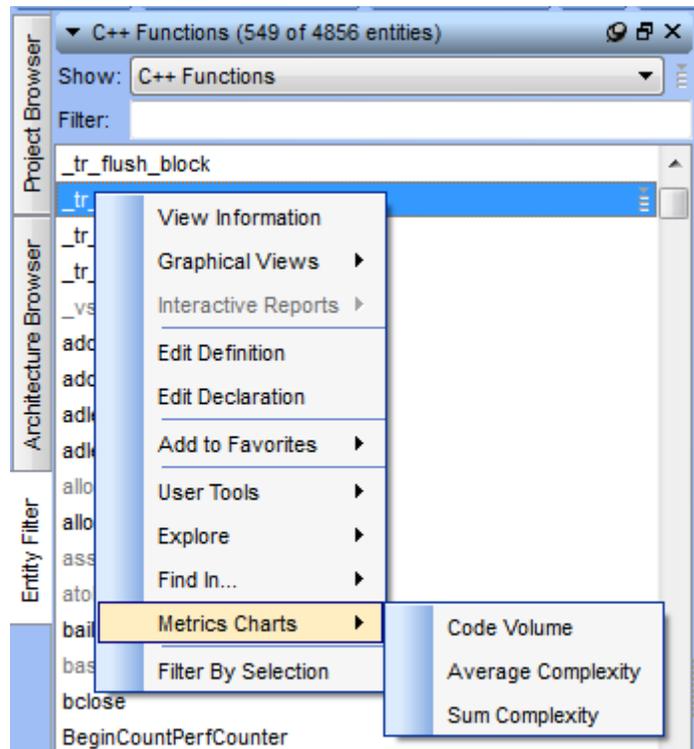
In this browser, select the following:

- **Architectures:** Check boxes next to one or more architecture nodes, files, and/or entities. The graph will provide a set of vertical bars for each of the items you select.
- **Metrics:** Select the type of metrics you want to graph from the drop-down list.
 - **Code Volume:** Provides a stacked vertical bar chart showing the count of lines that are blank, contain declarations and executable code, and contain comments.
 - **File Volume:** Provides a vertical bar chart showing the number of files in the selected architecture node that are code files vs. header files (or the number of files for languages that do not have header files).
 - **Average Complexity:** Provides a vertical bar chart of the average and maximum cyclomatic complexity for all nested functions or methods in the architecture node, along with the maximum nesting level of control constructs in the node's files.
 - **Sum Complexity:** Provides a vertical bar chart showing the number of possible paths through the code and the sum of the cyclomatic complexity and the essential complexity of all nested functions or methods.

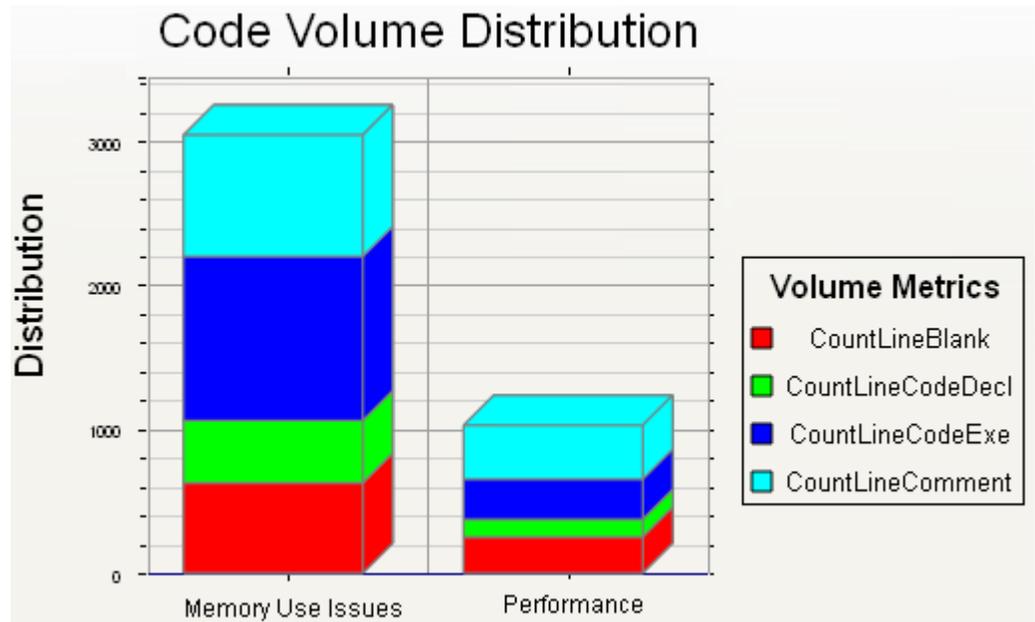
Click the **View Chart** button to display a chart for your selections or **View Table** to see the values in a table. If you have already selected a graph for this type of metrics, that tab will be reused unless you check the **Open in a new tab** box.



You can open metrics charts for various entities, including files and functions, from the context menus throughout *Understand*.



A typical metrics chart looks similar to the following:



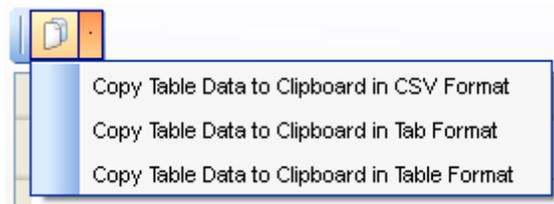
In a graph, you can choose the **Graph View** or the **Table View**. Both views have a toolbar that lets you save the graph or data.

In the **Graph View**, you can use the toolbar to:



- Save the image as a PNG, JPEG, or BMP image.
- Copy the image to the clipboard.
- Print the image using the standard Print dialog.

In the **Table View**, the numeric values for each pie slice or vertical bar is shown in a table. You can use the toolbar to copy the data to the clipboard in comma-separated (CSV), tab-separated, or table format (spaces used so columns align with headings if a font such as Courier is used).



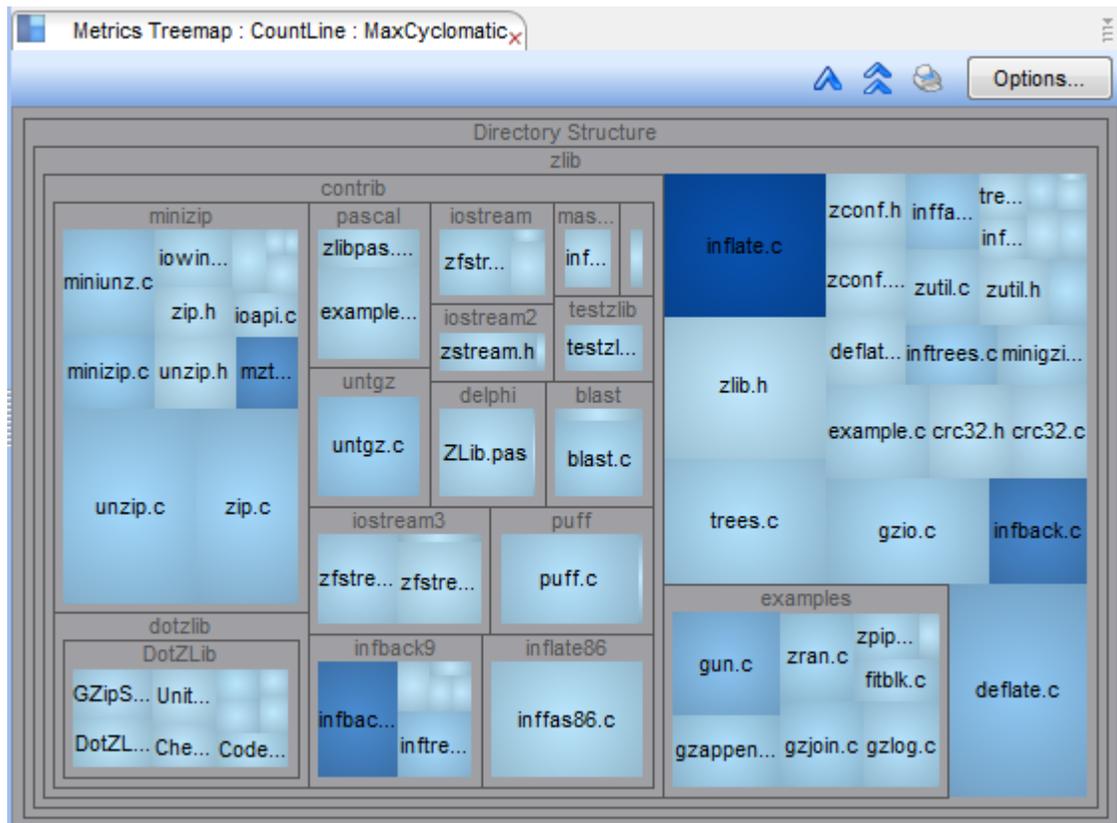
If you are viewing data for several architecture nodes, you can change the number in the lower-right corner to the number of vertical bars you want to view on each page and click the checkmark icon. Then use the arrows in the corners to move from page to page. The text shows which vertical bars are currently shown out of the total number. For example, the figure below indicates that bars 3 and 4 out of a total of 5 are currently shown.



Using the Metrics Treemap

Treemaps show metrics graphically by varying the size of node blocks and the color gradient. Each node block represents a code file. Different metrics can be tied to size and color to help you visualize aspects of the code.

For example, the following treemap ties the number of lines in each file to the size of the block and the MaxCyclomatic complexity metric to the darkness of the blue. This allows you to learn which files are large and complex vs. files that are large and relatively non-complex.

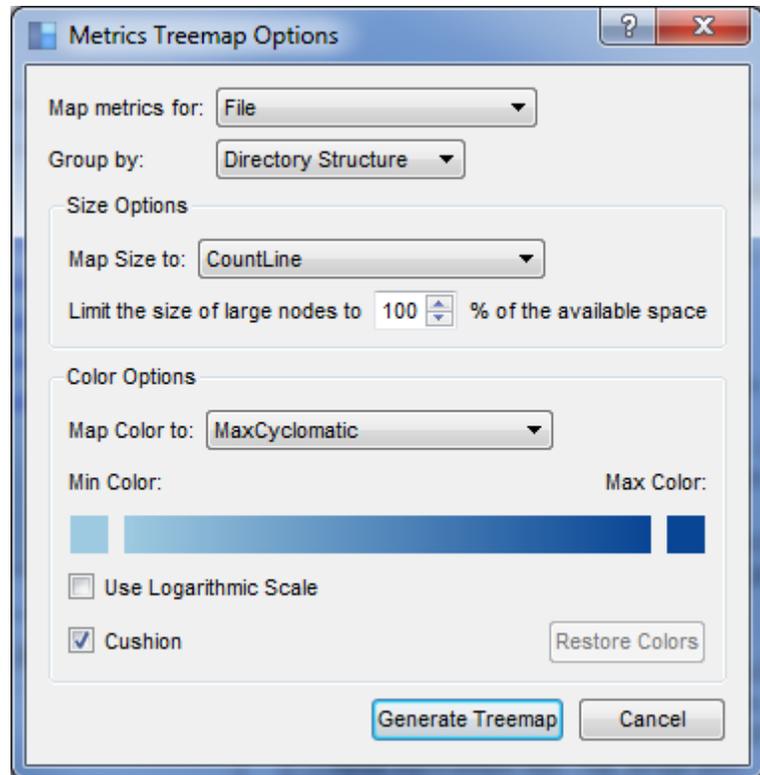


So we learn that `unzip.c` is large, but not particularly complex, while `inflate.c` is large and highly complex.

By default the maps are nested by directory structure. If you have built other architectures, you can use those as well.

To open the treemap for your project, follow these steps:

- 1 Choose **Metrics > Metrics Treemap** from the menus. You will see the Metrics Treemap Options dialog.



- 2 In the **Map metrics for** field, choose whether you want to select from metrics for Files, Classes/Interfaces/Structs, or Functions/Methods.
- 3 In the **Group by** field, choose how to group blocks in the treemap. The default is to group by the project's directory structure. Alternately, you can choose to group according to any other defined architecture or no architecture (flat).
- 4 In the **Size Options** area, choose a metric to control the size of the blocks. You can also limit the size of the largest blocks to some percentage of the treemap. You might want to use this if one node is taking up so much of the map that you can't see differences between the smaller nodes.
- 5 In the **Color Options** area, choose a metric to control block colors. You can click the left color square to set a color for blocks with the lowest value for this metric; click the right color square to set a color for blocks with the highest value for this metric.
- 6 Check the **Use Logarithmic Scale** box if you want the color scaled by powers of 10 of the selected metric value. This is useful for treemaps with extreme value ranges.
- 7 Uncheck the **Cushion** box if you want to see solid colors in the blocks. By default, the blocks have a gradient fill.
- 8 Click **Generate Treemap** to display the treemap. You can return to the Options dialog by clicking **Options** in the upper-right corner of the treemap.

Within the treemap, when your mouse cursor hovers over a block, the two metric values chosen for the size and color are shown.

You can double-click on an architecture node (shown as a gray border around a set of colored blocks) to display only the contents of that node. You can also zoom in by right-clicking on a node and choosing **Drill down** from the context menu.

After drilling down in the architecture, you can use the   icons to **Pop up one level** or **Pop up all levels** the treemap. You can also right-click to use the **Pop up one level** and **Pop up all levels** commands in the context menu.

The **Print** icon lets you print the treemap.

Exporting Dependency Metrics

The **Reports > Dependency** menu lets you export several types of files that provide metrics about dependencies between architectures, files, and classes/packages.

The output is for most of these commands is in CSV (comma-separated values) format, which can be opened with most spreadsheet programs. When you create a CSV file with *Understand*, it is automatically opened in a text file window.

The options available are as follows:

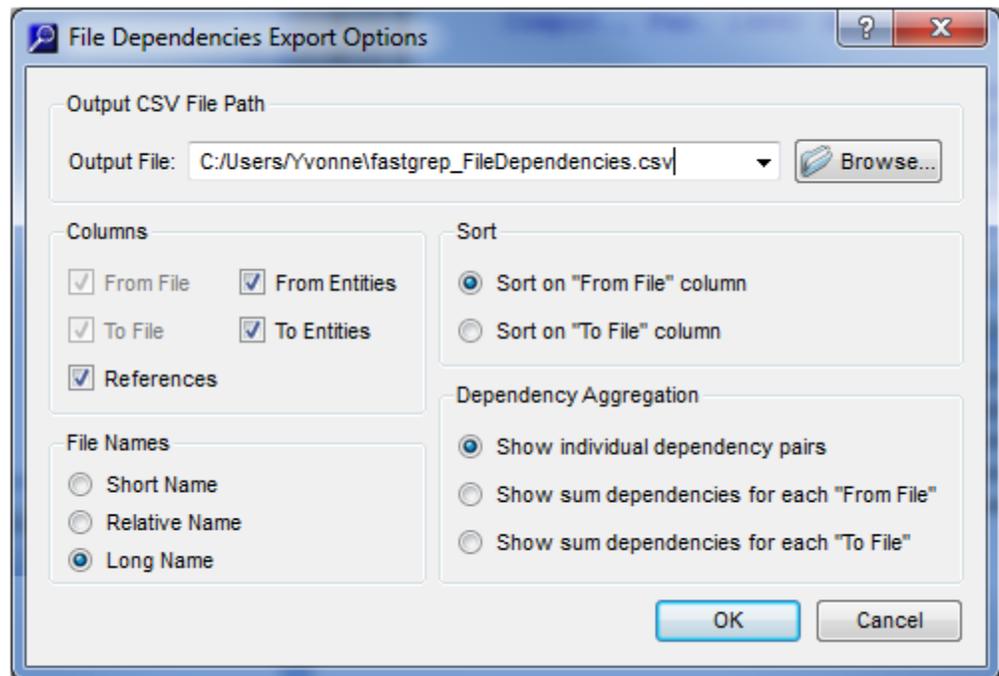
- **Architecture Dependencies >**
 - **Export CSV:** This output lists pairs of architecture nodes for which the node in column A is dependent upon the node in column B. The number of dependencies for each pair is listed in column C. (See page 226.)
 - **Export Matrix CSV:** This output lists all architecture nodes that are dependent upon others in column A. Row 1 lists all architecture nodes that are depended upon. The number of dependencies for each pair is listed at the appropriate row/column intersection. (See page 227.)
 - **Export Cytoscape XML:** This output format can be opened in Cytoscape (www.cytoscape.org), a free open-source program for analysis and visualization. It draws large diagrams very quickly, and can be useful if you want an overview picture of dependencies in a very large project. (See page 227.)
- **File Dependencies >**
 - **Export CSV:** This output lists pairs of files for which the file in column A is dependent upon the file in column B. The number of dependencies for each pair is listed in column C. (See page 226.)
 - **Export Matrix CSV:** This output lists all files that are dependent upon others in column A. Row 1 lists all files that are depended upon. The number of dependencies for each pair is listed at the appropriate row/column intersection. (See page 227.)
 - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See page 227.)

- **Class Dependencies >**

- **Export CSV:** This output lists pairs of classes and packages for which the entity in column A is dependent upon the entity in column B. The number of dependencies for each pair is listed in column C. (See page 226.)
- **Export Matrix CSV:** This output lists all classes and packages that are dependent upon others in column A. Row 1 lists all classes and packages that are depended upon. The number of dependencies for each pair is listed at the appropriate row/column intersection. (See page 227.)
- **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See page 227.)

Exporting Dependencies to a CSV File

When you choose to export a CSV file, you can also set the following options. (This figure shows the dialog for exporting File Dependencies; the other two CSV Export dialogs are very similar.)



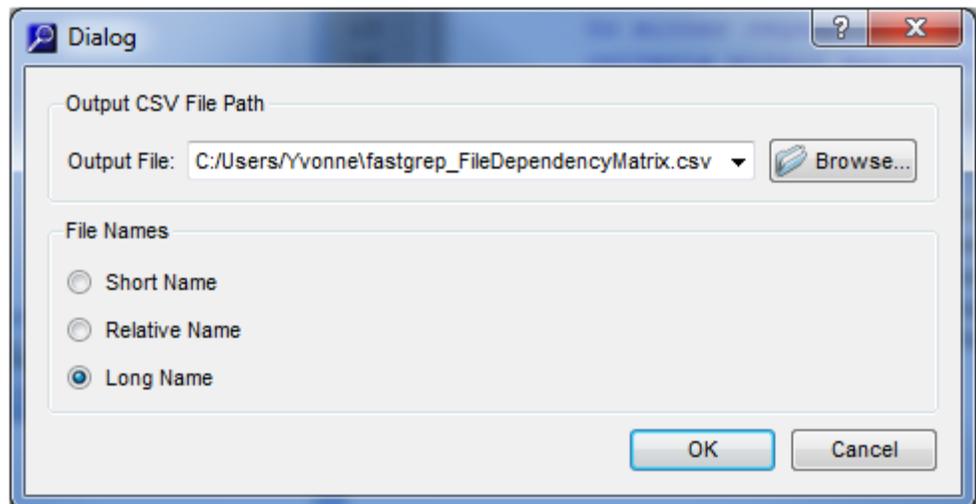
In this dialog, you can set the following options:

- **Select an architecture to analyze:** This option is available only when you are exporting architecture dependencies.
- **Output File:** Browse for the location to save the CSV file.
- **Columns:** Check the boxes for columns you want to include in the output. The “From” and “To” columns for the type of entity you are exporting are required, and cannot be deselected.
- **Names:** Choose a length for entity names. For example, all types can have a short or long name. Files can also have a relative name.

- **Sort:** Choose how you want dependencies sorted. You can sort based on the “From” column or the “To” column.
- **Dependency Aggregation:** Choose how you want to summarize dependency pairs that occur multiple times. You can show each pair individually or sum pairs for the “From” or “To” column for the type of entity you are exporting.

Exporting Dependencies to a CSV Matrix File

When you choose to export a CSV matrix file, you can also set the following options. (This figure shows the dialog for exporting File Dependencies; the other two CSV Export dialogs are very similar.)



In this dialog, you can set the following options:

- **Select an architecture to analyze:** This option is available only when you are exporting architecture dependencies.
- **Output File:** Browse for the location to save the CSV matrix file.
- **Names:** Choose a length for entity names. For example, all types can have a short or long name. Files can also have a relative name.

Exporting Dependencies to Cytoscape

Cytoscape (www.cytoscape.org) is an open source software tool for visualizing complex networks.

When you choose to export a Cytoscape file, you can Browse to select the location and filename for the output file. If you are exporting architecture dependencies, you can also select an architecture to analyze.

Once you have exported the *.xml file, you are asked if you want to open the file in Cytoscape. Note that you can only open Cytoscape if it is installed on your computer. See *Dependency Category* on page 104 for how to configure the location of the Cytoscape installation.

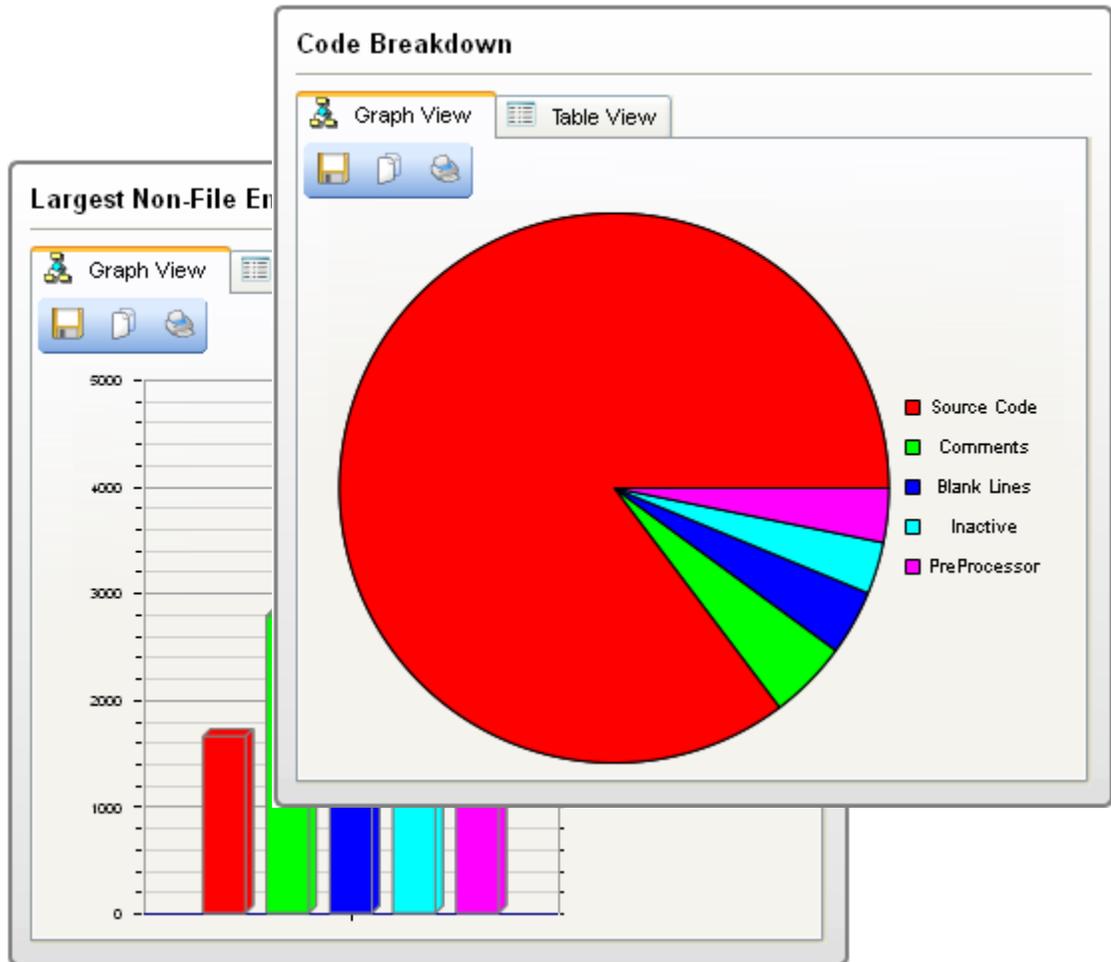
This chapter covers the graphical views in *Understand* and their options.

This chapter contains the following sections:

Section	Page
Project Overview Graphics	229
Graphical View Browsers	231
Types of Views	237
Graphical Notation	245
Controlling Graphical View Layout	245
Controlling Cluster Graph Layout	254
Saving Graphical Views	255
Printing Graphical Views	257

Project Overview Graphics

You can create graphics that provide an overview of your entire project by choosing **Project > Project Overview Charts** from the menus. This opens a tab in the document area that contains a number of pie charts and vertical bar charts. For example:



The graphs provided are code breakdown (line types), function breakdown, class breakdown, most complex functions, largest non-file entities, largest files, largest functions, best comment-to-code ratio entities, and most complex files.

In each area, you can choose the Graph View or the Table View. Both views have a toolbar that lets you save the graph or data.

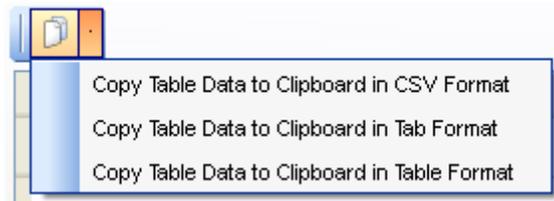
In the Graph View, you can use the toolbar to:



- Save the image as a PNG, JPEG, or BMP image.
- Copy the image to the clipboard.

- Print the image using the standard Print dialog.
- Zoom in on the graph in a new tab.

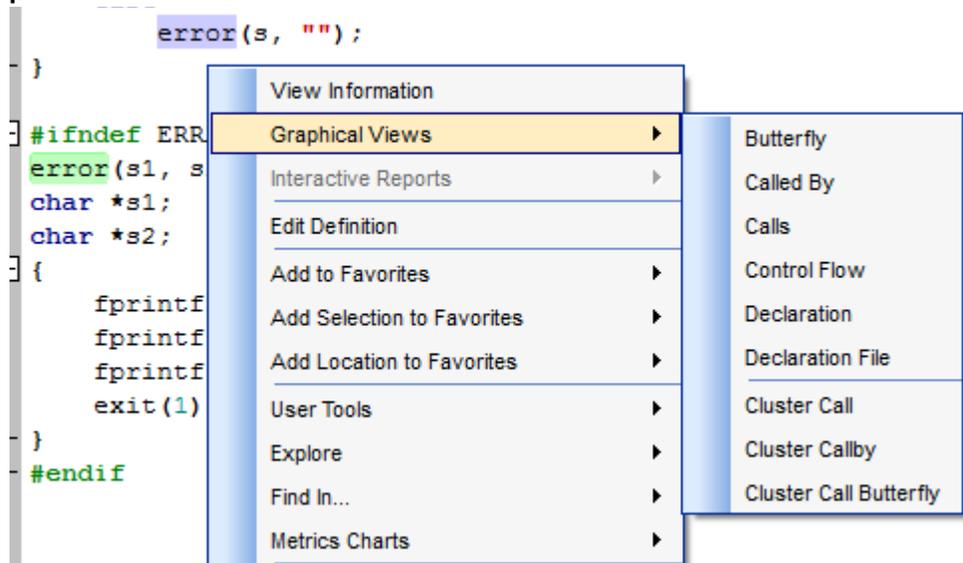
In the Table view, the numeric values for each pie slice or vertical bar is shown in a table. You can use the toolbar to copy the data to the clipboard in comma-separated (CSV), tab-separated, or table format (spaces used so columns align with headings if a font such as Courier is used).



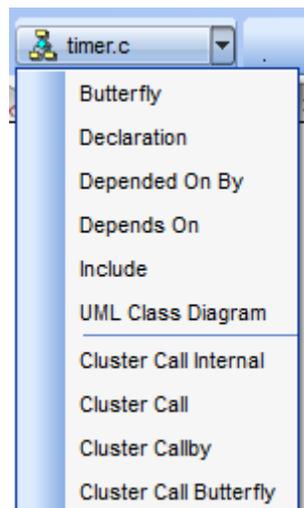
In addition to the Project Overview Charts, you can display metrics graphs that provide additional statistical information about your project or portions of your project. For details, see page 220.

Graphical View Browsers

The context menu of an entity that has a structure or hierarchy offers a choice called **Graphical Views**:



You can also use the Graphs drop-down menu in the toolbar to select from the types of graphs available for the entity at the current cursor position in a Source Editor tab. The same list of graphical views is available by choosing **Graphs > Graphs for <current_entity>** from the menus.

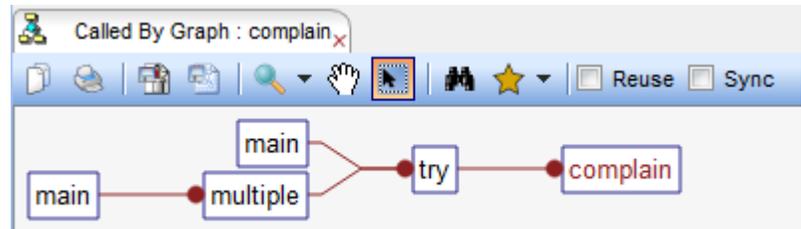


The **Graphical Views** menu adapts based on the kind of entity right-clicked. An item is greyed-out if information is normally available for this kind of entity but is not applicable to this particular entity (for instance a package that could be WITHed but isn't).

There are two main types of graphical views: hierarchy views and structure views.

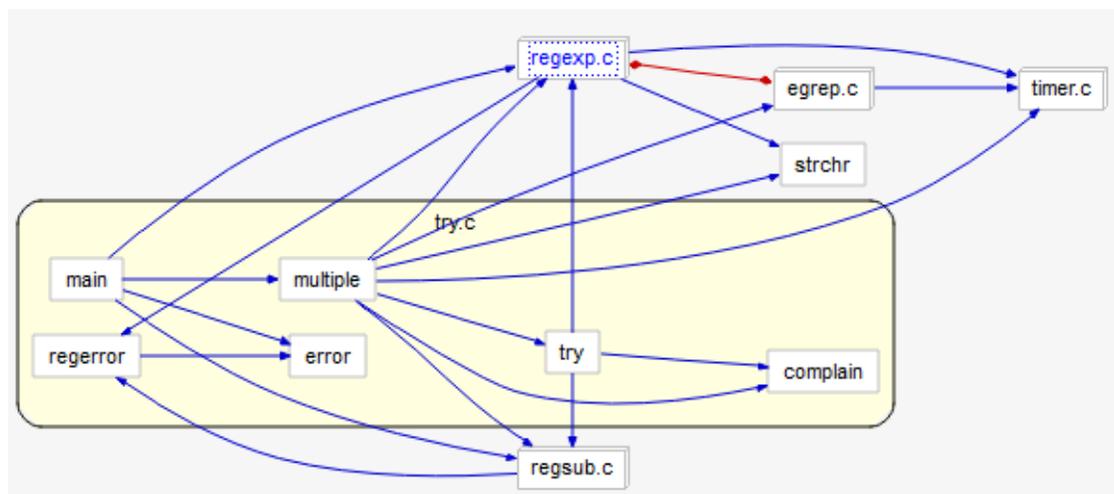
Hierarchy Views

A *hierarchy view* shows multiple level relationships between entities. All relationships are multi-level and are shown to the top or bottom of their respective tree unless a level option is set in the preferences. The following is a Call By graph for a function.



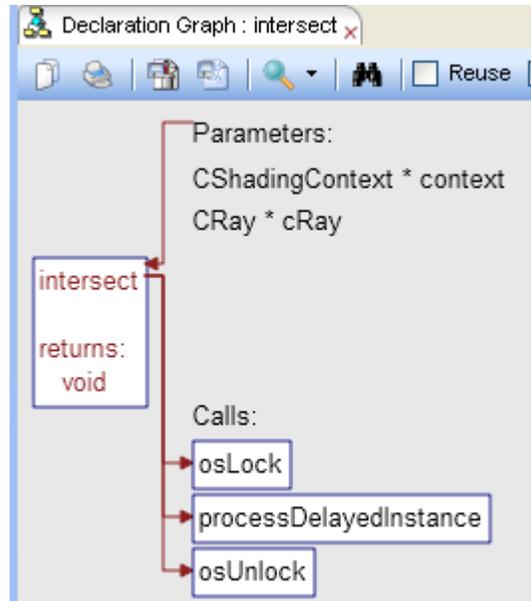
See *Hierarchy View Types* on page 237 and *Hierarchy View Examples* on page 238.

Cluster views are a special type of hierarchy view. They provide a more interactive view of call relationships. The Call, Callby, Butterfly and Internal Call variants are available, and can be accessed from the function, class, file, or architecture level. See *Controlling Cluster Graph Layout* on page 254.



Structure Views

Structure views offer a one glance way to see important structure and relational information about a given entity. The following is an example of a Declaration structure view:



See *Structure View Types* on page 240 and *Structure View Examples* on page 241.

General Commands for Using Graphical Browsers

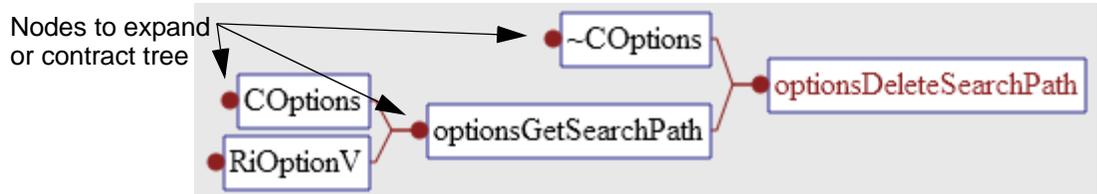
There are some general commands that can be used for browsing graphical views. Note that some of these tools are not available in all types of graphs.

- **Entity info:** Anywhere you see an entity, you can right-click on it to see a menu that offers many ways to learn more about that entity. Single-clicking shows information about the entity in the Information Browser. If you are in Screen Drag mode or Zoom mode, click the  icon to be able to select entities.
- **Searching:** Click the  **Search** icon at the top of a graphical view or press Ctrl+F to display the incremental search bar. You can use this bar the same way you use it in the Source Editor to find entities by name or other text in the current graphical view. As you type search text, all instances of the string are highlighted in the graphical view. See page 155 for details.

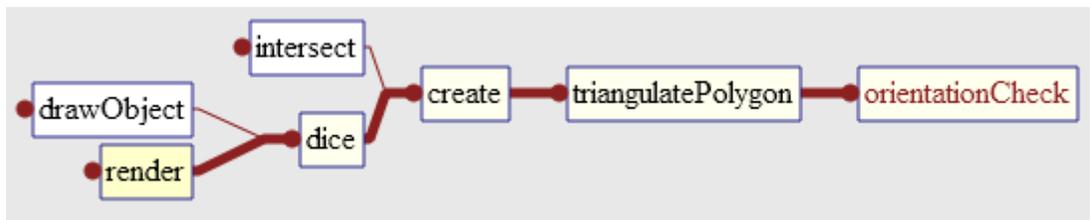


- **Opening source:** Right-click on an entity in a graphical view and choose **Edit Definition** to open the source location where the entity is declared.
- **Listing open views:** You can choose **Window > Windows** from the menus or look at the tabs across the top of the document area to see a list of all the separate graphical views you have open.
- **Scrolling:** You can scroll around a graphical view by dragging your cursor within the view when you have selected Screen Drag Mode by clicking the  icon.

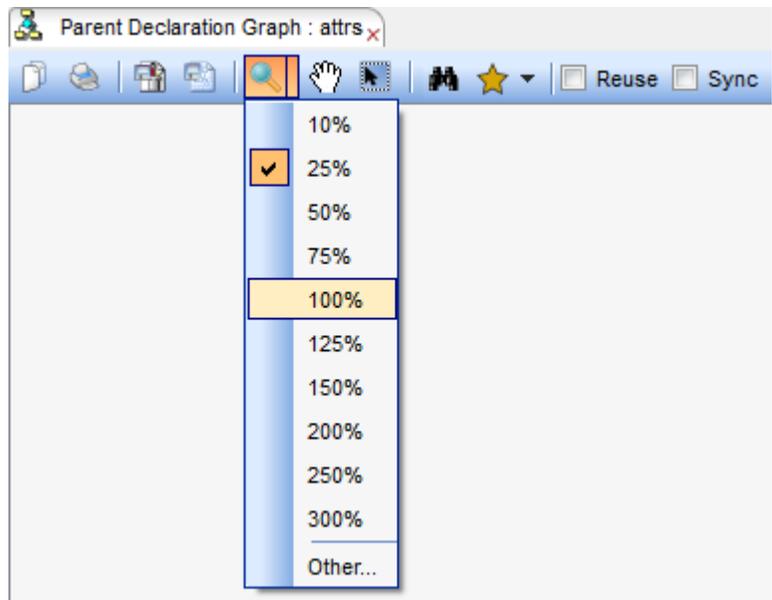
- **Expanding hierarchy:** You can expand and contract tree views by clicking the red circle to the right of a node. Right-click on the background of a view and choose **Open All Nodes** or **Close All Nodes** to expand or contract all nodes at once.



- **Path highlighting:** To highlight the path for a particular entity in a tree view (such as a Callby view), select the entity and right-click. In the context menu, choose **Highlight Path**.



- **Add to favorites:** You can add graphs you may want to reopen to a favorites list by clicking the  Favorites icon in the toolbar.
- **Zooming:** You can zoom in or out using the toolbar.



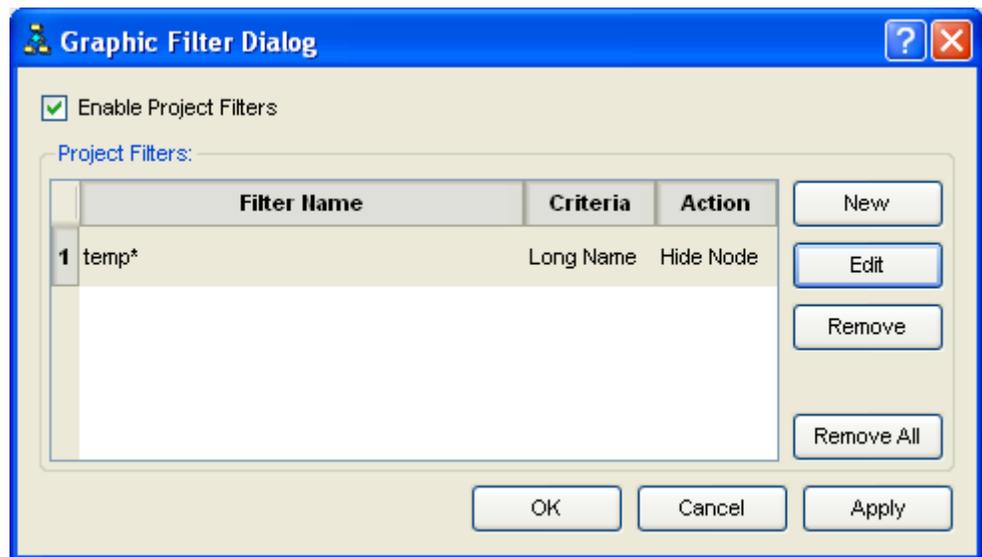
- **Printing and saving:** Everything you see can be printed or saved. Printing may be done to one page (squeezing the picture) or across multiple pages (poster style). See *Printing Graphical Views* on page 257 for details on printing. Graphical views can be saved as BMP, JPEG, PNG, Visio XML, and DOT files. See *Saving Graphical Views* on page 255 for details on saving to a graph file.

- **Layout control:** Layout is done automatically, there is no need to move lines or boxes around for a better view. Options are available for changing the layout. For example, you can control whether entities are sorted according to their order in the code or alphabetically. See *Controlling Graphical View Layout* on page 245.

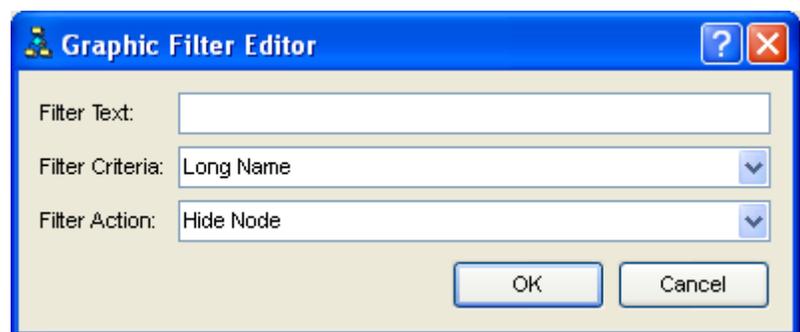
Filtering Out Entities

You can apply filters to hide certain entities in some graphical views. To create such a filter, follow these steps:

- 1 Right-click on the background of a graphical view and choose **Edit Graphic Filters** from the context menu. (Note that this option is not available for some types of graphs. For example, it is available for Call graphs and Declaration graphs.)



- 2 In the Graphic Filter dialog, put a checkmark in the **Enable Project Filters** box.
- 3 Click **New**. This opens the Graphic Filter Editor dialog.



- 4 Type a filter in the **Filter Text** field. For example, use `gr*` to match entity names beginning with `gr`. Filters are case-sensitive.
- 5 In the **Filter Criteria** field, select whether to compare the filter to long names, definition files, or the type text of entities. For example, if you choose long names, a filter of `print*` does not match `SomeProc::printWide`. Instead, you can type `*print*`.
- 6 In the Action field, select one of the following options:

- **Hide Node:** Items that match the filter are not included in the output.
- **Hide Sub Nodes:** The item that matches the filter is shown, but any subnodes of these items are removed from the output.
- **Collapse Sub Nodes:** Any subnodes of items that match the filter are collapsed in the output. An icon is shown after the node to indicate that there are subnodes. Items that match the filter are shown.

7 Click **OK** to add the filter to the project.

You can also create filters by right-clicking on an entity in a graphical view and choosing one of the filtering options. The options allow you to quickly filter out entities with that name or in that file.

You can remove filters you have created by clicking **Remove** or **Remove All**.

The filters you create apply to all graphical views that support filtering. You can temporarily disable filtering in the Graphical Settings dialog or by right-clicking on any graphical view and choosing **Disable Graphic Filters** from the context menu.

Reuse Checkbox

The **Reuse** checkbox controls whether a view is reused or a new window is opened when another graphical view is requested. The Reuse box is unchecked by default. At most one graphical view can have the **Reuse** box checked at any time.

You can cause views to be reused if a similar type of graphical view is opened from within a graphical view, no matter whether the **Reuse** box is checked. Change this behavior in the User Interface > Windows category of the **Tools > Options** dialog (page 94).

Sync Checkbox

The **Sync** checkbox controls whether this graphical view changes when a different entity is selected in the Project Browser, Entity Filter, and other windows that let you select an entity. For example, if you check the Sync box in a Declaration graph window and then select a different entity in the Entity Filter, the graph shows declaration information for the newly selected entity.

Graph Options

See page 113 for information about the **Graphs** category in the **Tools > Options** dialog. You can control how the display of relationships between graph nodes changes when you hover the mouse over a graph or double-click on a node.

Types of Views

There are two main types of graphical views: hierarchy views and structure views.

Hierarchy View Types

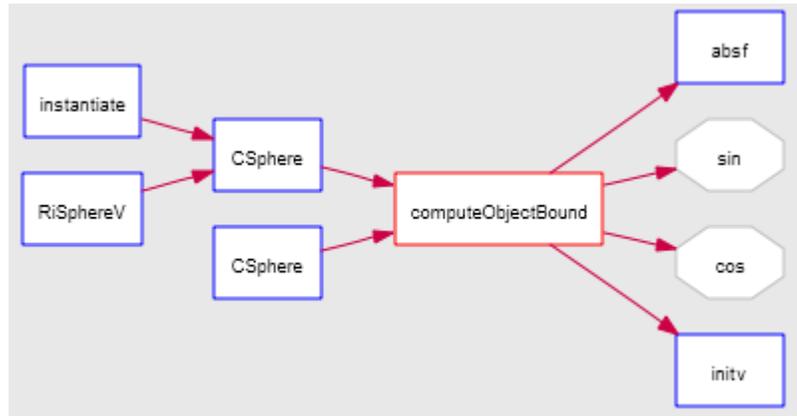
Hierarchical views show multi-level relationships between entities. *Understand* offers hierarchy graphs of the following types of relationships. Some types apply to specific source languages.

- **Butterfly:** Shows both calls and called by.
- **Calls:** Shows who this entity calls.
- **Called By:** Shows who calls a given entity.
- **Include:** Shows who this file includes.
- **IncludeBy:** Shows who includes this file.
- **Depends On Graph, Depended On By Graph, and Butterfly Graph:** Available for classes, packages, and architectures only. See page 179 for architecture graphs.
- **Derived Classes:** Shows classes derived from a given class.
- **Base Classes:** Show what classes are the base for a class.
- **Extends:** Shows which classes extend this class.
- **Extended By:** Shows which classes are extended by this class.
- **Class Inheritance:** Shows who inherits from a given class.
- **Child Lib Units:** Shows Child Library Units of a compilation unit. (Ada 95 only)
- **Declared In:** Shows the declaration tree from where this program unit is declared.
- **Declaration Tree:** Shows declaration nesting of program units in a compilation unit.
- **Instantiated From:** Shows instantiation tree of a generic type or compilation unit.
- **Instantiations:** Shows who instantiates a given generic unit.
- **Invocation:** Shows what compilation units a unit invokes.
- **Parent Lib Unit:** Shows the parent lib units of a given entity.
- **Type Derived From:** Shows tree of types a type is derived from.
- **Type Tree:** Shows types that derive new types from an entity.
- **With:** Shows what compilation unit an entity “Withs” into scope.
- **WithBy:** Shows what compilation units “Withs” a given entity.
- **Uses:** Shows which modules use this item.
- **Used By:** Shows which modules are used by this item.
- **Cluster Call Internal:** Shows call relationships within a file.
- **Cluster Call:** Shows who this entity calls.
- **Cluster Callby:** Shows who calls this entity.
- **Cluster Butterfly:** Shows both calls and called by.

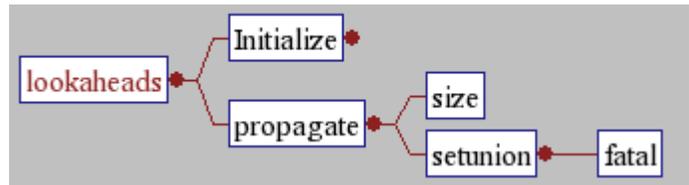
Hierarchy View Examples

Hierarchy views show multi-level relationships between entities. Here are examples of the types of hierarchy views that *Understand* offers.

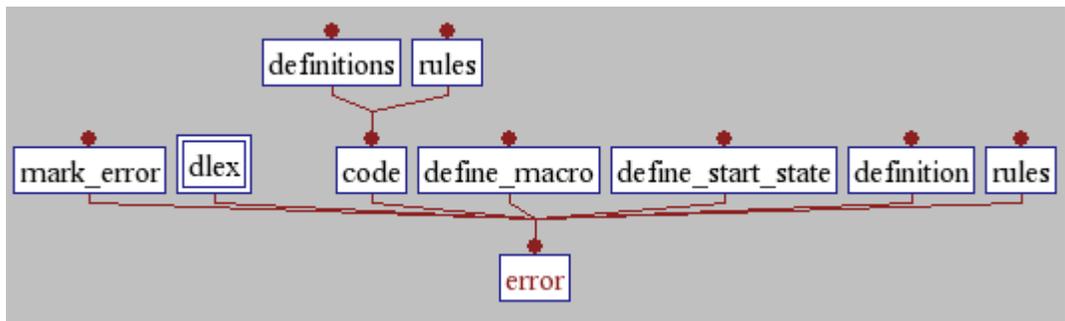
- **Butterfly:** Shows both calls and called by relationships if they exist. The selected entity is outlined in red.



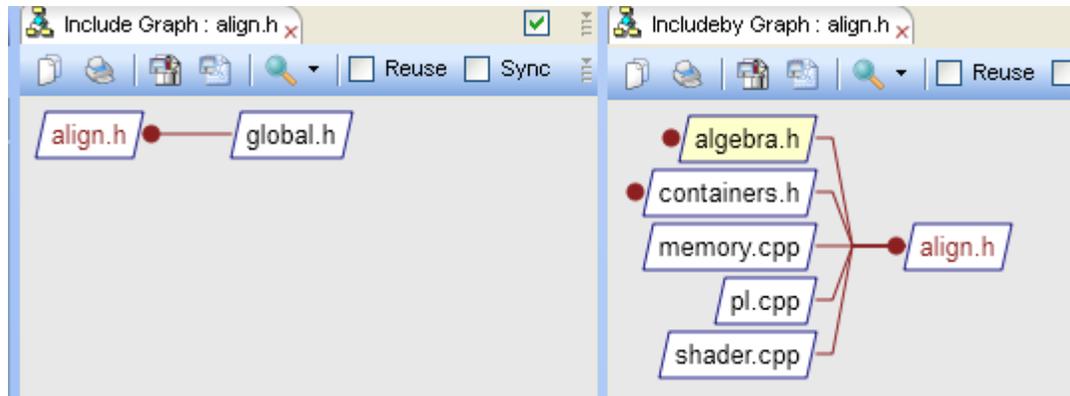
- **Calls:** Shows the entire chain of calls emanating from this function. Each line between entities is read as “x calls y”.



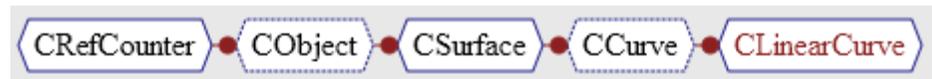
- **Called By:** Shows what calls an entity. Each line connecting an entity is read as “x is called by y”. In this example, error is called by code (and others), which is called by rules (and others). Note that this view is read from the bottom up or right to left.



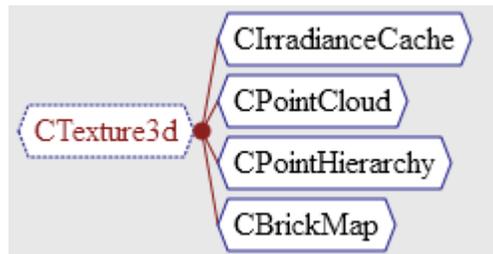
- **Include:** Shows the include hierarchy of an entity, such as a file. A connecting line is read as “x includes y.” In this example, align.h includes global.h.



- **Include By:** Shows the include tree in the other direction. In the previous example, align.h is included by several files such as algebra.h.
- **Base Classes:** For classes, shows the base classes from which this class is derived from. In this example, class *CLinearCurve* is derived from class *CCurve*, which is derived from class *CSurface* and so on.



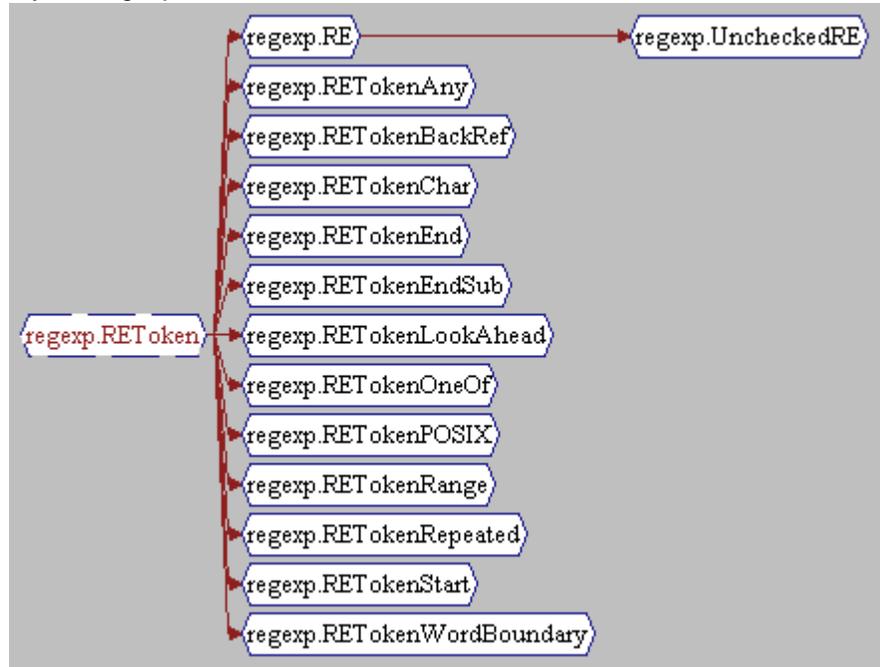
- **Derived Classes:** Shows the classes that are derived from this class. In this example, class *CTexture3d* is a base class for classes *CIrradianceCache* and others.



- **Extends:** Shows which classes extend other classes. In this example, the *regexp.UncheckedRE* class extends the *regexp.RE* class, which extends the *regexp.REToken* class.



- **Extended By:** Shows which classes are extended by other classes. A line is read as “class is extended by class.” In this example, the `regex.REToken` class is extended by a number of classes, including the `regex.RE` class, which in turn is extended by the `regex.UncheckedRE` class.



Structure View Types

Structure views offer a one glance way to see important structure and relational information about a given entity. Understand structure views include the following:

- **Architecture Graph:** Shows the hierarchy of an architecture node. See page 179.
- **Declaration:** Shows what a structure is composed of. For example, shows the parameters, return type, and callbys of a function. For classes, shows what members are provided, who inherits this class, and who it is based on.
- **Parent Declaration:** Shows what a structure is composed of. Shows Calls instead of the Called Bys shown by a Declaration graph.
- **Declaration File:** Shows what entities (such as functions, types, macros, and variables) are defined within a given file.
- **Declaration Type:** Shows what a type is composed of.
- **Class Declaration:** Shows the members defining the class and the parent class
- **Data Members:** Shows what components a class, struct, or type contains.
- **Control Flow:** Shows a flow chart of the function or similar entity type. Clicking on a node in the graphs jumps to the line of code referenced.
- **UML Class Diagram:** Shows the classes defined in the project or a file and related classes. Adheres to the Unified Modeling Language (UML) structure diagram format.
- **Package:** Shows what entities are declared in a given package (body or spec).

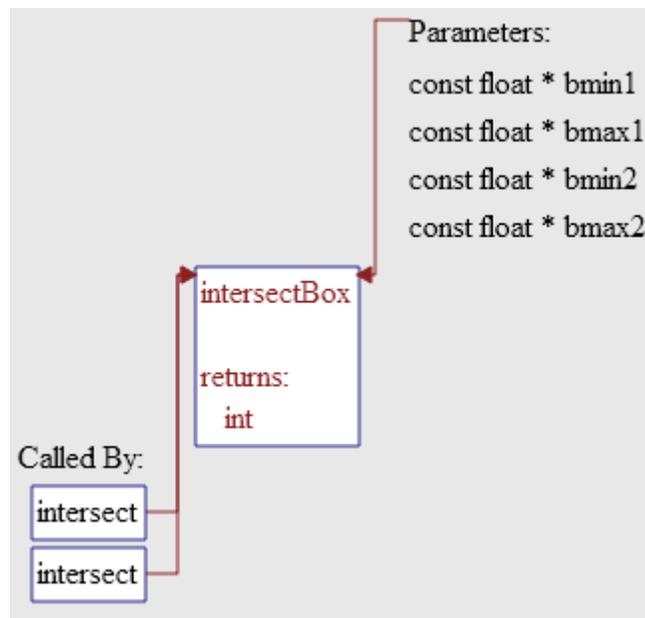
- **Task:** Shows the parameters, invocations, and what entities/entry points are declared in a task. Also shows what the task Withs.
- **Rename Declaration:** Shows what entities are renamed in the entity.

Structure View Examples

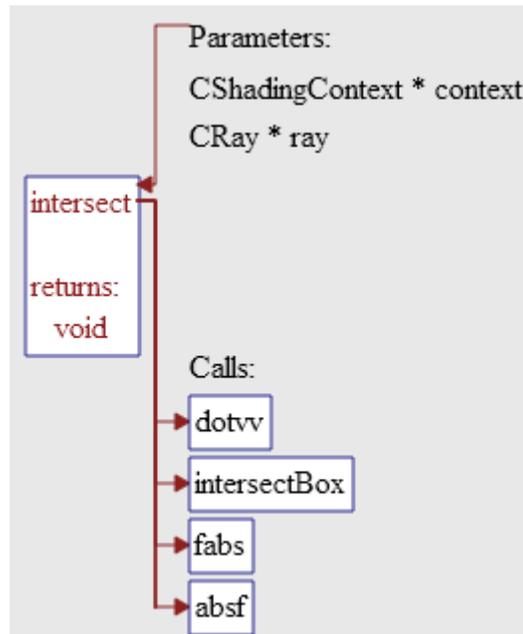
Structure views quickly show structure and relations.

Understand structure views are designed to present essential information about an entity in a small and concise manner. The structure diagram is derived from the graphs presented by Booch and Buhr in their respective books “Software Engineering with Ada” and “System Design in Ada.” Where needed, symbols and annotations have been extended or altered to represent new kinds of information available from *Understand*.

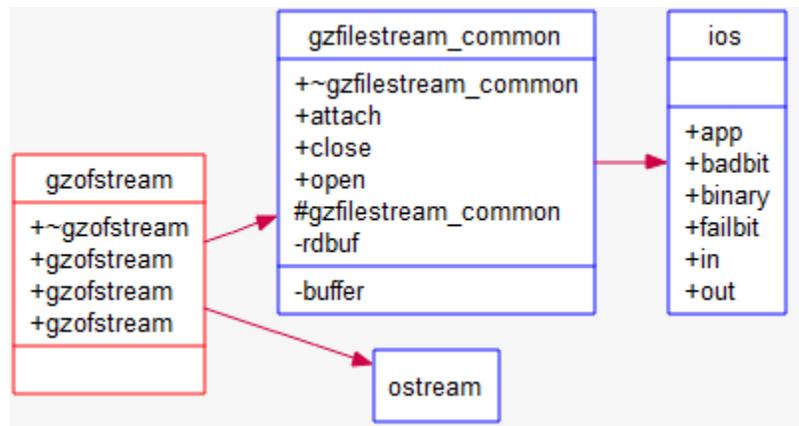
- **Declaration:** Shows the structure of the entity. For example, shows the parameters, return type, and callbys of a function.



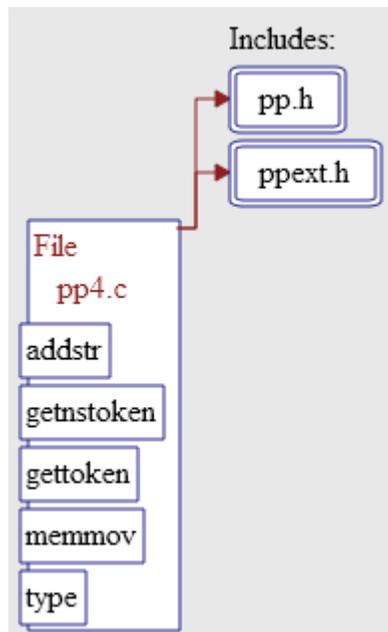
- **Parent Declaration:** Similar to a Declaration graph but shows what the entity calls.



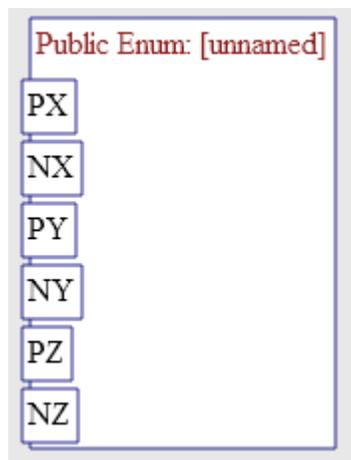
- **UML Class Diagram:** Shows the classes defined in the project or a file and related classes. Right-click to show or hide class details, related classes, and solo classes.



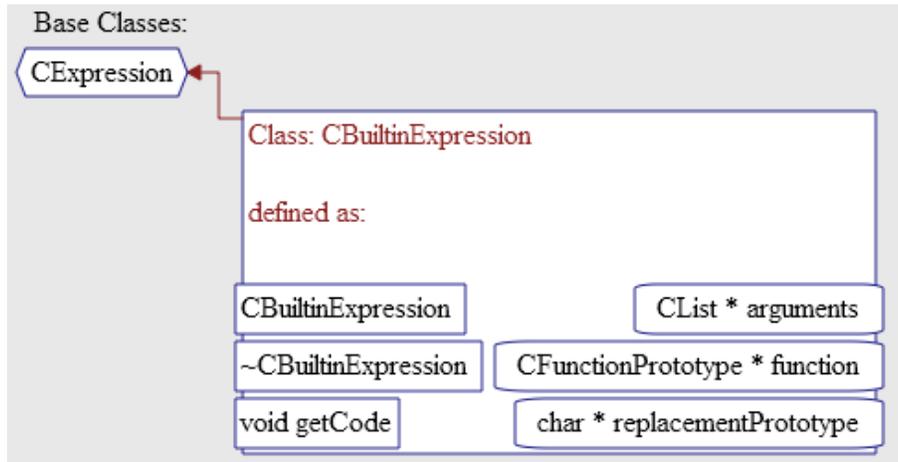
- **Declaration File:** Shows the entities declared in the file. Also shows files included by the file and classes imported by the file.



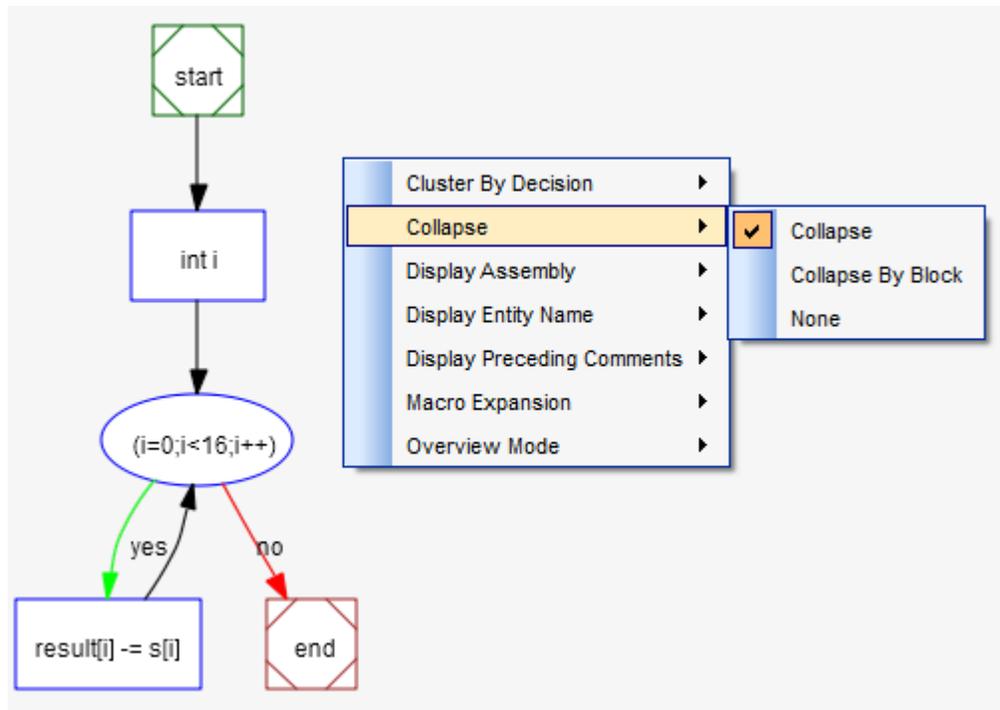
- **Declaration Type:** Shows information about a type declaration.



- **Class Declaration:** Shows the members defining the class and the parent class from which it is derived.



- **Control Flow:** Shows a flow chart of the function or similar entity type. As the following figure shows, a number of specialized options can be set when you right-click on this type of graph.



Graphical Notation

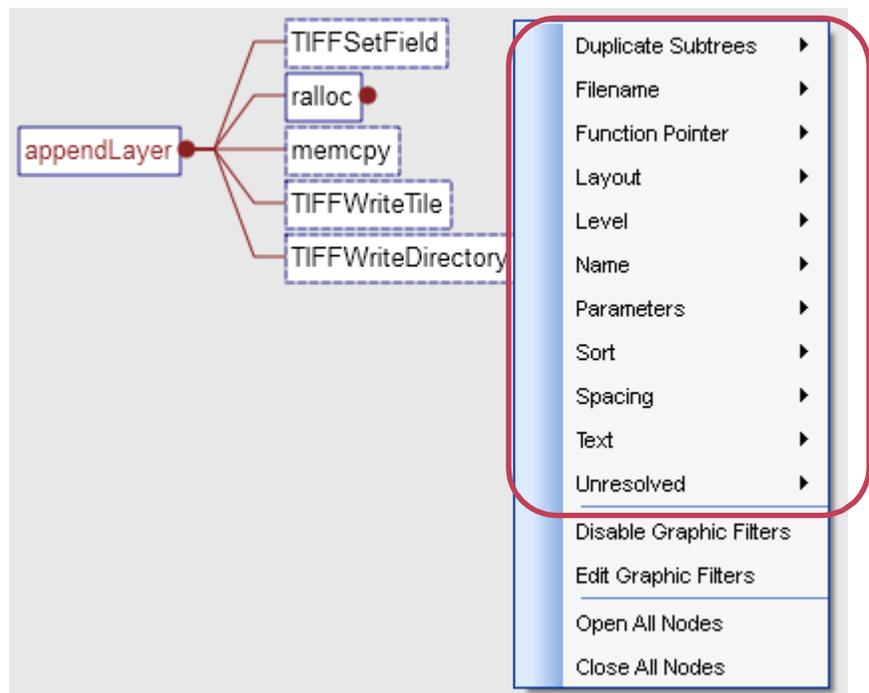
The following symbols are used by *Understand* to represent various language constructs. The symbols vary somewhat depending upon the type of view.

- Entities such as functions and other program units are shown in rectangles.
- Files and system-level entities are usually shown in parallelograms.
- Classes and types are shown in flattened hexagons.
- Macros are usually shown in flattened octagons.
- Objects such as variables are usually shown in slightly rounded rectangles.
- Unknown or unresolved entities are drawn with dashed outlines or in gray.
- Other shapes are language-specific.

In Control Flow views, standard flow chart symbols, such as diamonds for decision points, are used.

Controlling Graphical View Layout

The two main types of graphical view windows, **Hierarchy** and **Structure**, have a variety of configuration options. You can set them by right-clicking on the background of a graphical view and choosing the option you want to modify from the context menu.



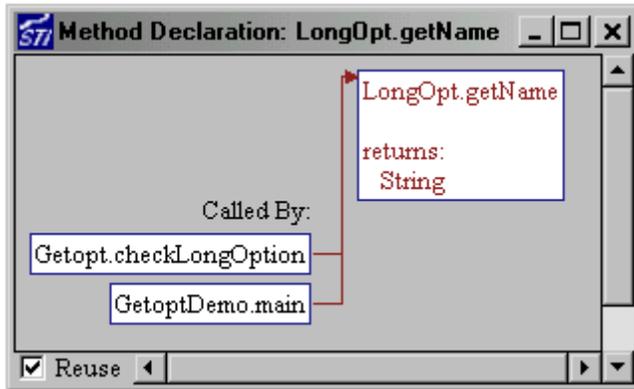
These options control the layout and drawing of the graphic views and vary based on the current type of view. The following subsections describe a number of these options.

Note that the options for Cluster graphs are different, and are described in *Controlling Cluster Graph Layout* on page 254.

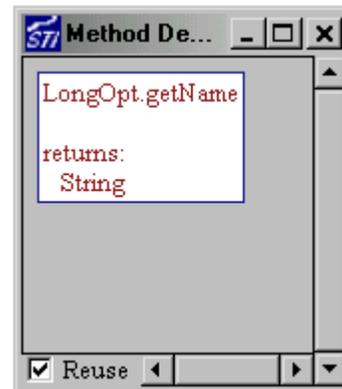
Called by Menu

The **Called by** menu controls whether program units that call the current entity are shown in declaration views.

View with Called By set to On



View with Called By set to Off



Constants Menu

The **Constants** menu controls whether to show constants in Declaration views. The default is On.

Default Members Menu

The **Default Members** menu controls whether declaration views show default members of the class.

Dependent Of Menu

The **Dependent Of** menu controls whether files a C file is dependent on are drawn in the C File Declaration view. The Default is On.

Dependent Menu

If **Dependents** is on (the default) then files dependent on the current C file are shown in a File Declaration view.

Depth

Sets the number of levels to which a dependency graph is expanded. The default is 1 level.

Duplicate Subtrees Menu

The **Duplicate Subtrees** menu controls whether multiple occurrences of the same subtree are shown in hierarchy views. The options are to Hide or Show such subtrees. The default is to show duplicate subtrees. In some applications, hiding duplicate subtrees can dramatically simplify hierarchy views. Duplicate subtrees are not shown if a view has over 1000 nodes.

Expand Recursive Notes Controls whether recursive nodes in a dependency graph are shown as separate items. The default is to show the expansion for recursive nodes. If you turn this setting off, a particular item is expanded only at the highest level where it occurs in the architecture, class, or package hierarchy. Unexpanded nodes that are recursive at lower levels display “(recursive)” as part of their text.

Expand Repeated Notes Controls whether repeated nodes in a dependency graph are shown as separate items. The default is to show the expansion for repeated nodes. If you turn this setting off, a particular item is expanded only at the highest level where it occurs in the architecture, class, or package hierarchy. Unexpanded nodes that are repeated at lower levels display “(repeated)” as part of their text.

Extended By Menu The **Extended By** menu controls whether declaration views show classes by which the selected class is extended.

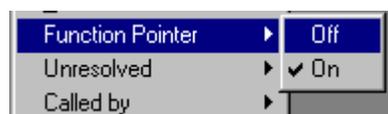
Extends Menu The **Extends** menu controls whether declaration views show classes that the selected class extends.

External Functions Menu If **External Functions** is on then functions defined in a header file or in a file included by a header file are shown in the Declaration View for a header file. Default is On.

Filename Menu The **Filename** menu controls how filenames are displayed in views. It is available for both declaration and hierarchy views. The options are Off and On.:

- **None:** Filenames are not shown in the view.
- **Shortname:** Where filenames are relevant, only the name of the file is shown in square brackets.
- **Fullname:** Where filenames are relevant, the full file path and filename are shown in square brackets.

Function Pointer Menu The **Function Pointer** menu controls whether function pointers are displayed as invocations in the Call and CallBy trees.



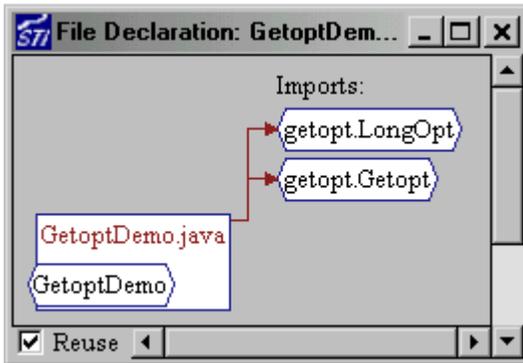
Globals Menu The **Globals** menu controls whether to show globals in Declaration views. The default is On.

Implements Menu The **Implements** menu controls whether declaration views show entities that the selected entity implements.

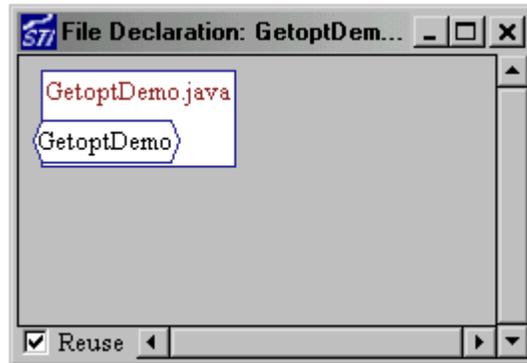
Implemented By Menu The **Implemented By** menu controls whether declaration views show entities by which the selected entity is implemented.

Imports Menu The **Imports** menu controls whether declaration views show entities imported by the current entity.

View with Imports set to On



View with Imports set to Off



Included By Menu If **IncludeBy** is on (default) then files that include the Header File being drawn in a Header File Declaration view are shown.

Includes Menu The **Includes** menu controls if include files are drawn on file declaration diagrams (C file, Header file). Default is On.

Inherits Menu The **Inherits** menu controls whether declaration views show entities that the selected entity inherits.

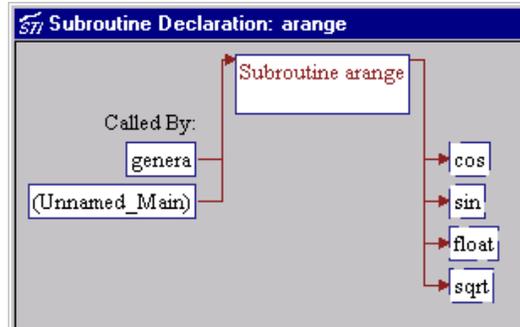
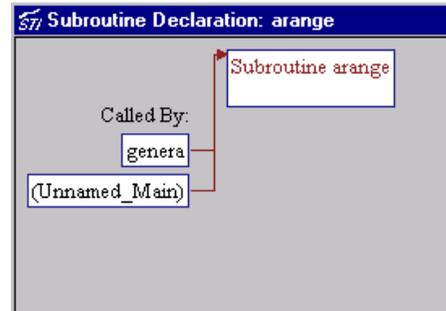
Inherited By Menu The **Inherited By** menu controls whether declaration views show entities inherited by the selected entity.

Intrinsic Menu The **Intrinsic** menu controls whether intrinsic functions (for example, cos and sin) are displayed or hidden.



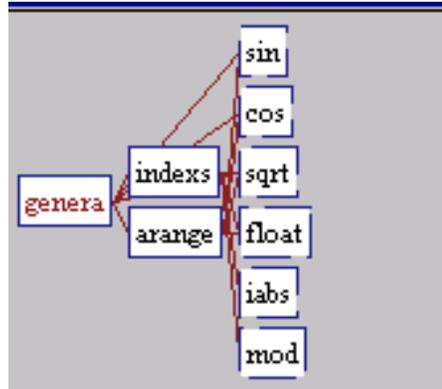
Invocations Menu

The **Invocations** menu controls whether procedures and functions called by the current procedure or function are shown in Declaration views.

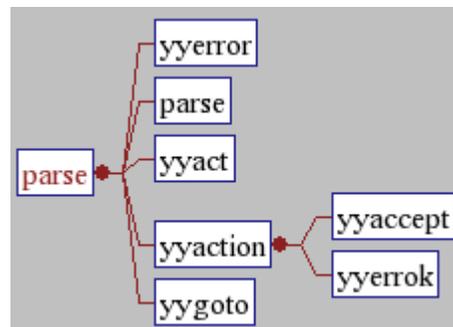
View shows Invocations**View without Invocations shown****Layout Menu**

The **Layout** menu controls the layout algorithm for a hierarchical chart. It is available only in hierarchy views (calls, callby, etc.). The options are:

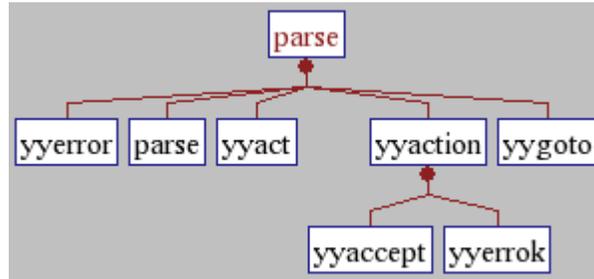
- **Crossing:** A left-to-right view, minimizing space used but sacrificing some readability by permitting lines between entities to cross.



- **Horizontal Non-Crossing:** A left-to-right layout, using more space in some situations but enhancing readability by having no crossing lines.

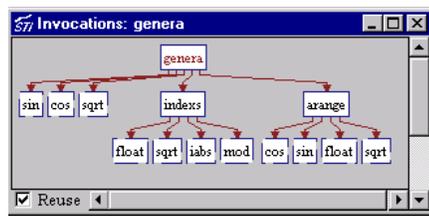


- **Vertical Non-Crossing:** A top-to-bottom layout similar to Horizontal Non-Crossing.

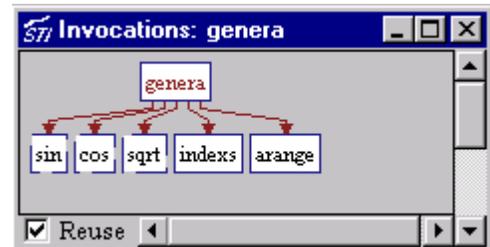


Level Menu

The **Level** menu controls the number of levels to be traversed when laying out a hierarchical view. The default value is “All Levels”. Values of 1 to 5 may be set. It is available only in hierarchy views.



All Levels



One Level

Locals Menu

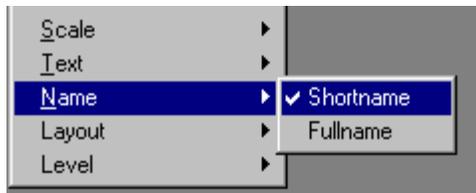
The **Locals** menu controls whether local items are shown in Declaration views. The default is On.

Members Menu

The **Members** menu controls whether members and operators are shown in the Type Tree and Type Derived From views. The choices are to show None, Components, Operators, or Operators and Components.

Name Menu

The **Name** menu controls whether or not fullnames are used in views. It is available for both declaration and hierarchy views.



A fullname includes its parent compilation units. For example:

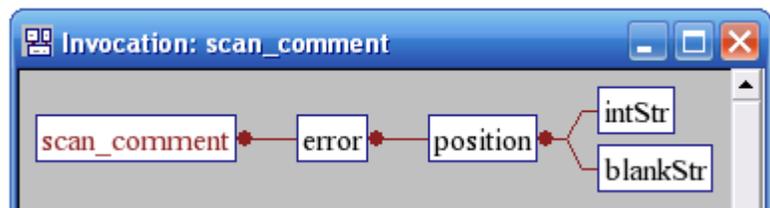
- Text_lo.Put is the fully specified name.
- Put is the Short Name

Longer versus shorter names can alter the layout of pictures substantially.

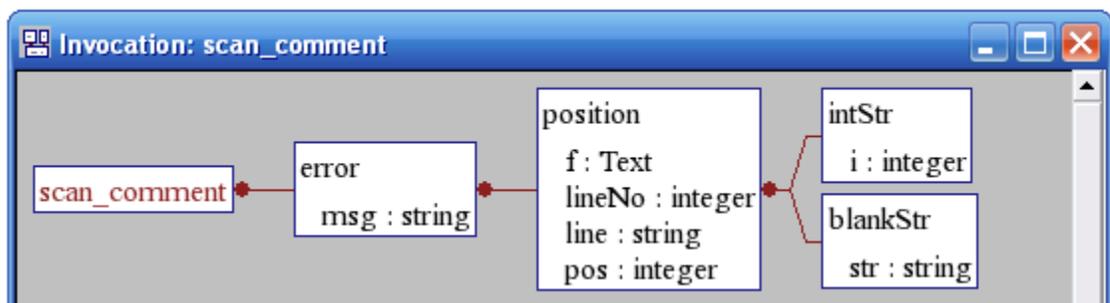
Objects Menu The **Objects** menu controls whether to show objects in Declaration views. The default is On.

Operators Menu The **Operators** menu controls whether entities that are operators are shown in the Callby, Declaration, Declaration Tree, and Invocation views.

Parameters Menu The **Parameters** menu controls whether parameters are shown in hierarchical views. Available on any hierarchical graphical view (invocation and callby). The default is Off, turning this On can make hierarchical pictures much bigger.



Parameters Off (the default)



Parameters On

Private Members Menu The **Private Members** menu controls whether declaration views show private members of the entity.

Protected Members Menu The **Protected Members** menu controls whether declaration views show protected members of the entity.

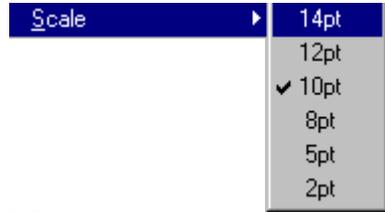
Public Members Menu The **Public Members** menu controls whether declaration views show public members of the entity.

Renames Menu The **Renames** menu controls whether declarations that are renames are shown in Declaration views. The default is to show rename declarations.

Routines Menu The **Routines** menu controls whether to show routines (procedures, functions, ...) in Declaration views. The default is On.

Scale Menu

The **Scale** menu allows you to choose the size of the text used. It is available for both declaration and hierarchy views. All picture sizes and layouts vary with text point size. The currently selected size is indicated by a check mark.



Other point sizes can be added by customizing configuration files found in the *Understand* installation directory. Contact support@scitools.com for information on how to do this.

Sort Menu

The **Sort** menu lets you specify whether entity names in tree views should be sorted alphabetically. If this option is off (the default), entities are sorted in the order they are encountered in the project.

Spacing Menu

The **Spacing** menu lets you choose to change the space between boxes. You can choose compact, small, normal, wide, or extra wide.

Sql Menu

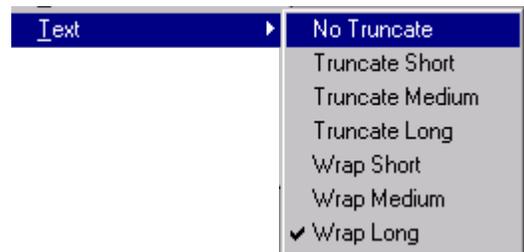
The **Sql** menu lets you specify whether SQL entities should be shown in graphical views. This option is on by default.

Static Menu

The **Static** menu controls if static functions are drawn in function, C File and Header File declaration views. Static functions are those declared using the “static” keyword. They are visible only within the file they are declared in. If enabled static functions are drawn with the edge of their box inside the edge of the outer declaration box for their enclosing unit (C file). Default is On.

Text Menu

The **Text** menu sets the way entity names are trimmed or altered to accommodate the layout of graphics. It is available for both declaration and hierarchy views. Names may be truncated to a certain length or wrapped at a certain length.



- **No Truncation:** Uses the name as defined in the source code. The default.
- **Truncate Short:** Cuts off names at 10 characters.
- **Truncate Medium:** Cuts off names at 20 characters.
- **Truncate Long:** Cuts off names at 30 characters.

- **No Wrap:** Never wraps text to the next line.
- **Wrap Short:** Wraps the name between 8 and 10 characters. Location in that range depends on if a natural wrapping character is found. Natural wrapping characters are . _ - and :
- **Wrap Medium:** Similar to *Wrap Short* except wrapping range is 15-20 characters.
- **Wrap Long:** Similar to *Wrap Short* except wrapping range is 20-30 characters.

Types Menu

The **Types** menu controls whether to show types in Program Declaration views. The default is On.

Typetext Menu

The **Typetext** menu tells declaration views (Function Declaration, C File Declaration, Header File Declaration) to include types on the view. Default is On.

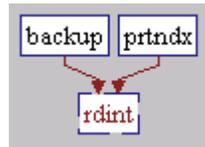
Unknown Menu

The **Unknown** menu controls whether entities that are used, but for which no declaration was found should be drawn. Unknown methods and entities are those used in the analyzed source without a definition in the same source.

Unresolved Menu

The **Unresolved** menu controls whether entities that have been used but no declaration was found should be drawn. This option is available on hierarchy and structure views. Unresolved functions and entities are those used in the analyzed source without a definition in the same source. Unresolved include files are those included but not found along a declared include path (either a compiler or project include path).

Unresolved entities are drawn as normal but with a dashed border:

**Usedby Menu**

The **Usedby** menu tells Declaration views whether to show items that use this item.

Uses Menu

The **Uses** menu tells Uses views whether to show only items that are used directly, or to also show items that are used by nested subprograms. The default is to show both.

Variables Menu

The **Variables** menu controls whether to show globals in Declaration views. The default is On.

Withs Menu

The **Withs** menu controls on Declaration views of compilation units (packages, tasks, separate procedures, etc...) if Withs are drawn. The default is On.

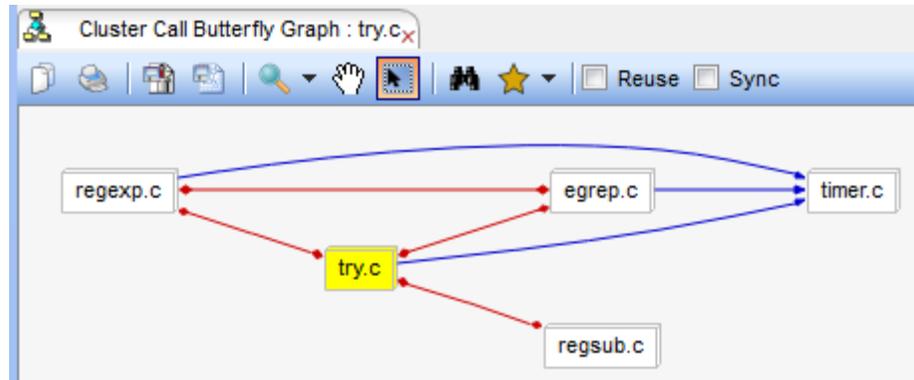
With Bys Menu

Controls if **With Bys** (who Withs a given compilation unit) are shown on Declaration views. The default is On.

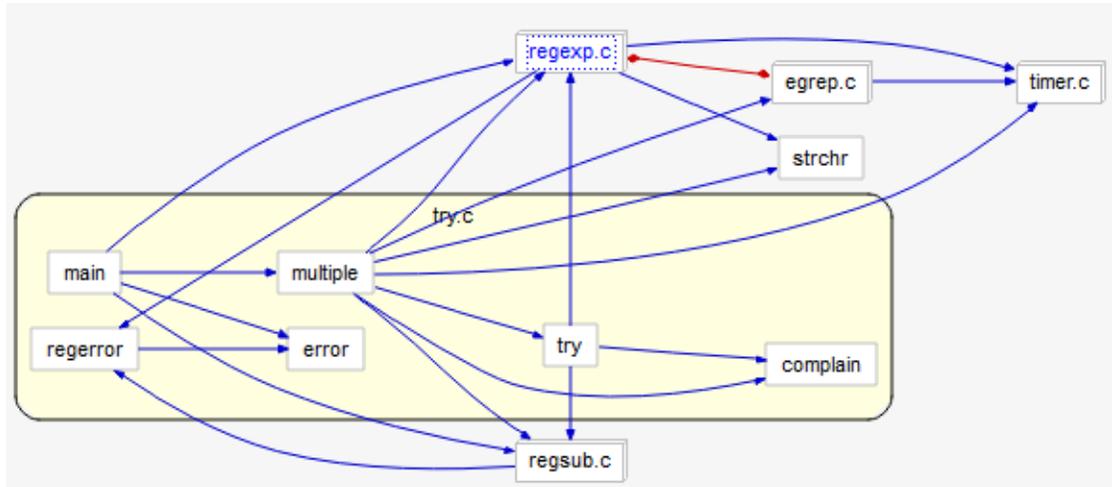
Controlling Cluster Graph Layout

Cluster graphs are a special type of hierarchy view. They provide a more interactive view of call relationships than other hierarchy views. The Call, Callby, Butterfly and Internal Call variants are available, and can be accessed from the function, class, file, or architecture level.

For example, if you open a Cluster Call Butterfly graph for a file, you see a graph similar to the following:



If you then double-click on some of the file boxes, you can see call relationships for functions within the files that you expand.



The toolbar for cluster graphs is the same as for other graphs, and the context menu for entities in the graph is similar to elsewhere.

The context menu when you click on the background of a cluster graph offers the following options:

- **Edges Shown:** Choose which relationships to the originally selected entity you want shown. “Forward” is call relationships. “Reverse” is callby relationships. “Butterfly” is both call and callby relationships.

- **Aggregate Nodes by:** Choose an architecture you want to organize entity nodes.
- **Entity Name Format as:** Choose whether you want to display short or long names for entities.
- **Show Edge Labels:** Check this item if you want the number of occurrences of this relation to be shown in the graph. For bi-directional call relationships, the two numbers in the label show calls in each direction.

The Graph Customizer to the right of a cluster graph offers the same settings as those described for Dependency Graphs in *Graph Customizer Toolbar* on page 180 and *Graph Customizer Fields* on page 180.

Saving Graphical Views

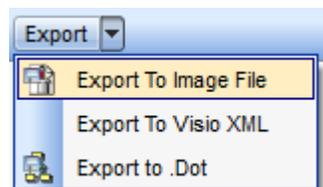
Understand offers a number of ways to export your graphical views and use them in other ways. The toolbar for each graphical view provides the following icons for copying and printing graphs.



In addition to printing, you can save graphical views as JPEG, PNG, SVG files (page 255), Visio XML files (page 256), and DOT files (page 256). The first three formats are common graphics formats.

Saving Views to Files

To save a graphical view in one of the following formats, use the **Export** drop-down the graphical view toolbar to choose the **Export to Image File** option. Or choose **File > Export to Image File** from the menus. In the Export dialog, choose a location, filename, and file type for the view.



- **JPEG** files are compressed bitmaps. They can be viewed with most web browsers, document editors, and graphics programs. This format is “lossy”; some data is lost in the compression.
- **PNG** files store compressed bitmaps similar to GIF files. They can be viewed with most web browsers, document editors, and graphics programs. They use a non-patented compression method.
- **SVG** files are Scalable Vector Graphics files. This file type uses XML to describe a 2-dimensional vector-based image.

You can also copy a graphical view to the clipboard and paste it as a bitmap into the image program or word processor of your choice. To do this, click the  **Copy** icon on the graphical view toolbar or choose **Edit > Copy Image to Clipboard** from the menus. Then, paste the image into another program.

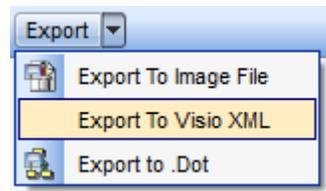
Note that if the graph would result in an image larger than 200 MB, the graph will be resized to a smaller size.

Saving Views as Visio Files

Microsoft Visio is a vector-based graphics program used for drawing flowcharts and similar graphics. That is, it deals with shapes and objects rather than pixels. Visio XML is an Extended Markup Language that is supported by Visio and a number of other graphics applications.

You do not need to have Visio installed in order to save a graphical view as a Visio XML file.

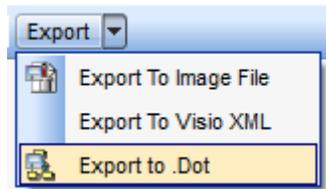
To save a Visio XML file, use the **Export** drop-down the graphical view toolbar to choose the **Export to Image File** option. In the Export dialog, choose a location and filename for the view. The file extension for Visio XML files is *.vdx.



Saving Views as DOT Files

DOT is a language used to describe graphs in plain text. This format can be imported and edited by a number of external tools. You can export many (but not all) types of graphs produced by *Understand* to a DOT file.

To save a DOT file, use the **Export** drop-down the graphical view toolbar to choose the **Export to .Dot** option. In the Export dialog, choose a location and filename for the view. The file extension is *.dot.



If this option is not shown in the **Export** drop-down, the current graph cannot be exported to the DOT format.

Printing Graphical Views

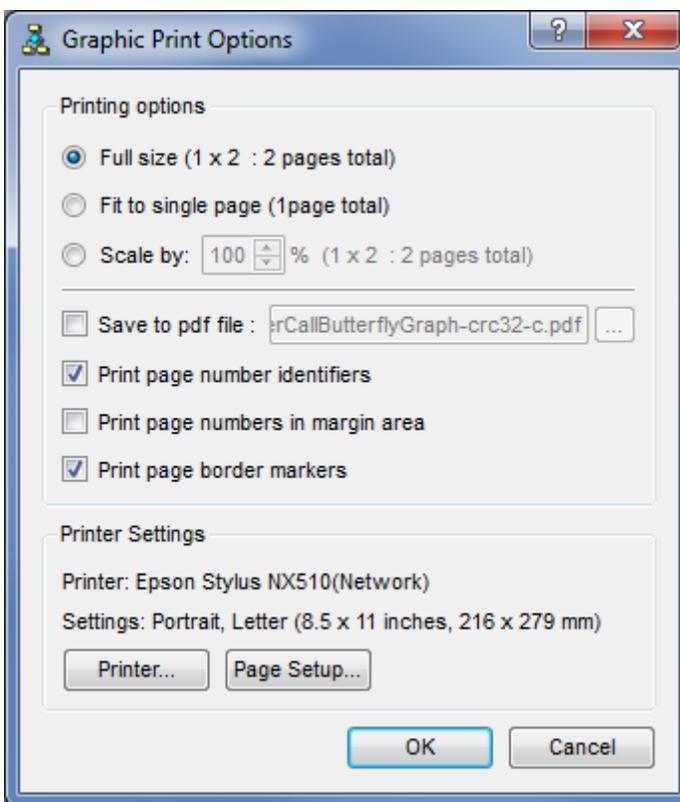
Understand has these printing modes:

- **Source File** printing sends a text file to the printer using 66 lines of source per page. See *Printing Source Views* on page 174.
- **Graphical view printing** provides options for how to fit the image to a page. See *Graphical View Printing* on page 257.

Graphical View Printing

To print the current graphical view, you can click  **Print** icon on the graphical view toolbar. Or, choose **File > Print Entity Graph** from the menus.

When you choose to print a graphical view, you see the Graphic Print Options dialog.



You can choose to print the image at one of the following sizes:

- **Full size** uses the default scaling of 100%. The dialog shows the number of pages in width x height format. The page size selected with Page Setup is used.
- **Fit to a single page** scales the image to fit on the selected page size.
- **Scale by** lets you choose the sizing percentage and shows the number of pages that will be printed.

Check the **Save to PDF** file box if you want the image saved to an Adobe Acrobat file rather than being sent to a printer. This PDF printing feature does not require that you have third-party PDF generating software installed on your computer.

Check the **Print page number identifiers** box if you want page numbers on each page in the upper-left and lower-right corners. The page numbers are in “(column, row)” format. For example, (1,3) indicates that the page goes in the leftmost (first) column of the third row when you piece the pages together. The page number is not printed if the view fits on a single page.



Check the **Print page numbers in margin area** to place the page numbers outside the borders of the graph. If this box is unchecked, page number indicators are printed just inside the border markers.

Check the **Print page border markers** box to place corner markers in each corner of each page.

Click the **Printer** button to open the standard Print dialog for your operating system. When you click **Print** or **OK** in that dialog, you return to the Graphic Print Options dialog.

Click the **Page Setup** button to open a Page Setup dialog, which allows you to choose the paper size, paper source (if applicable), page orientation, and margin width. Click **OK** to return to the Graphic Print Options dialog.

Click the **OK** button in the Graphic Print Options dialog to send the graphical view to the printer (or a PDF file).

Note: The **File > Page Setup** menu option applies only to printing source code and other text files. The **Page Setup** button on the Graphic Print Options dialog saves its settings separately.

Using CodeCheck for Standards Verification

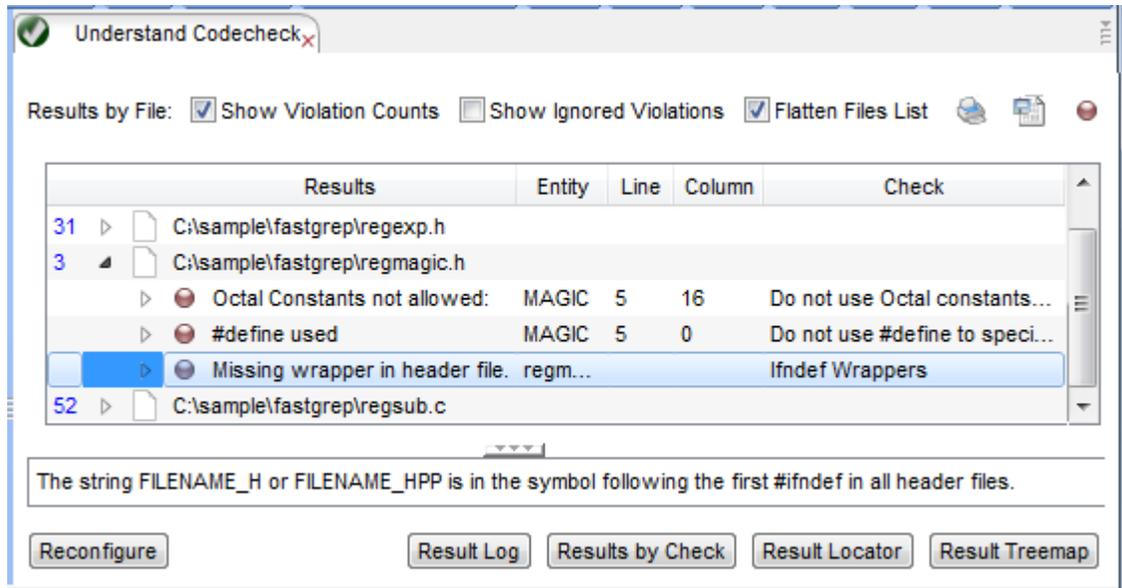
This chapter explains how to use CodeCheck to find places where your code does not conform to standards you select.

This chapter contains the following sections:

Section	Page
About CodeCheck	260
Running a CodeCheck	261
Viewing CodeCheck Results	264
Using CodeCheck Configurations	271
Writing CodeCheck Scripts	272

About CodeCheck

Understand provides a tool called CodeCheck to make sure your code conforms to published coding standards or your own custom standards. These checks can be used to verify naming guidelines, metric requirements, published best practices, or any other rules or conventions that are important for your team.



Checks are available to make sure your code conforms to several published coding standards. You can select a subset of individual checks to test for from these standards. For example, you can check to make sure that all if...elseif constructs contain a final else clause.

For all languages, checks are provided to let you verify that various entity types conform to your naming conventions and to confirm that your code meets metric requirements you set for complexity, function length, and nesting depth.

If you want to perform custom checks, you can create your own checks using Perl. For example, you can create a check to find lines longer than 80 characters or filenames that begin with a number.

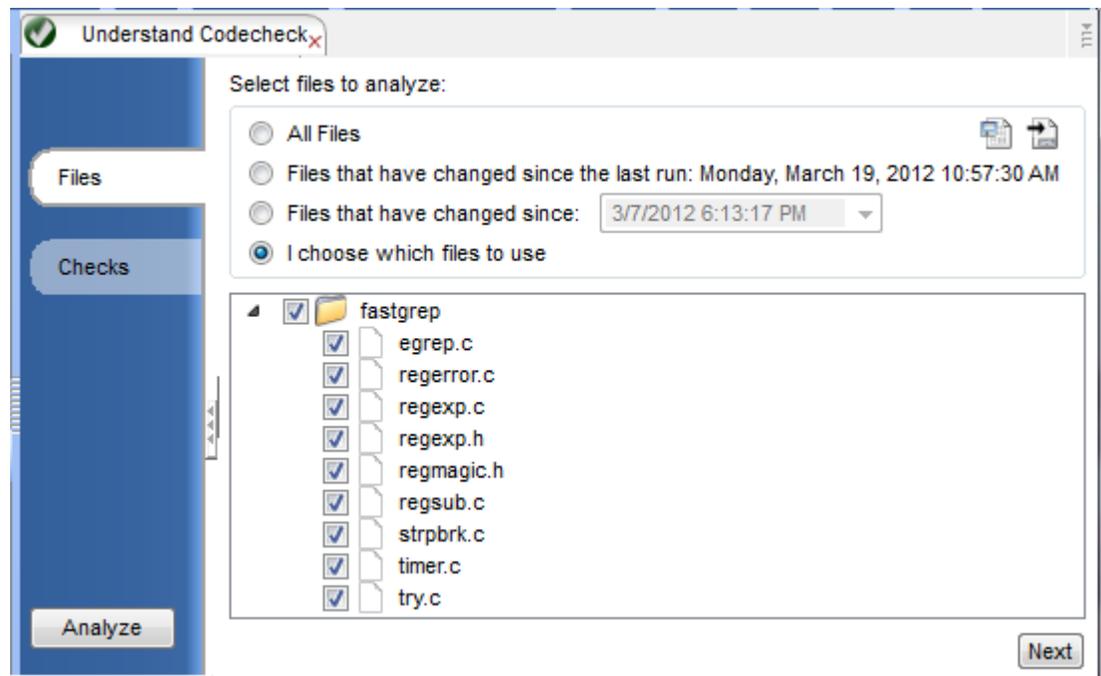
CodeCheck validation suites are available from the [SciTools Blog](#).

Running a CodeCheck

To open the CodeCheck tool, choose **CodeCheck > Open CodeCheck** from the menus.

Files Tab

In the **Files** tab, choose whether to check all files in the project, only files that have changed since you last ran CodeCheck, only files that have changed since a specific date, or the files you select.



If no files have been changed since the date you select or since the last time you ran CodeCheck, you will see a message that says no files meet the criteria.

If your project contains many files, you can more easily select specific files by following these steps:

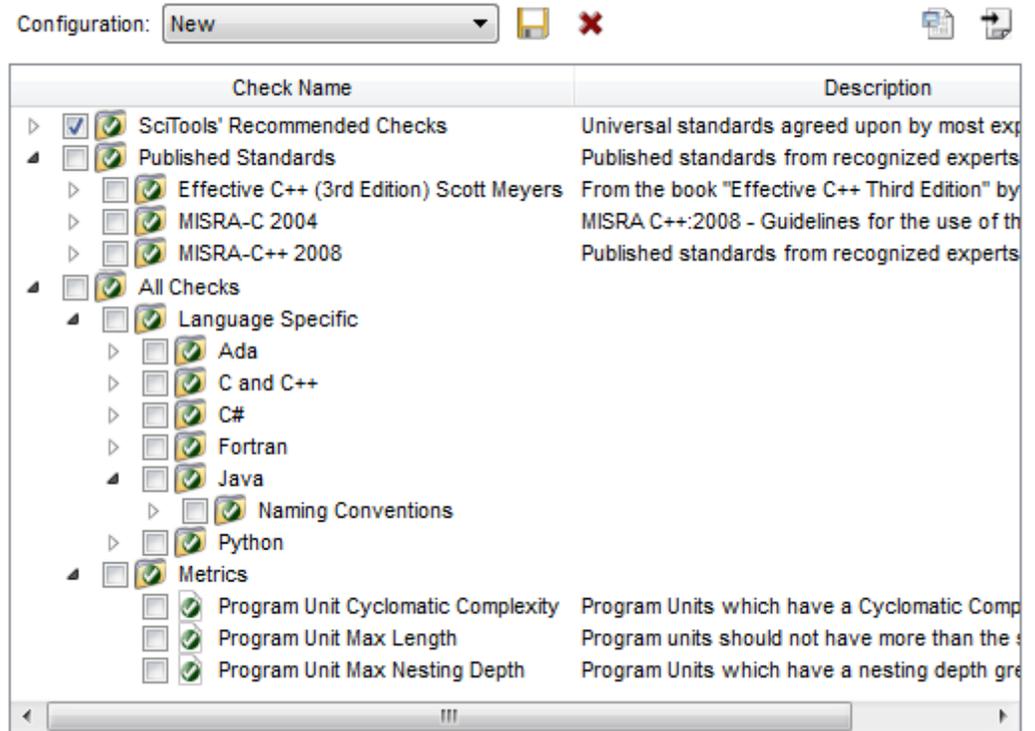
- 1 Export the entire list of currently selected files (with full file paths) to a text file by clicking the  **Export Selected Files** icon.
- 2 Edit the file with a text editor. Delete the lines for files you do not want to analyze, and save the file as plain text.
- 3 Click the  **Import File List** icon and import the file you edited. This will select only the files listed in the file.

Once you have finished selecting the files to analyze, click **Next**.

Checks Tab

In the **Checks** tab, the default configuration is **New** and all the **SciTools' Recommended Checks** that apply to your programming languages are selected.

If you want to use a different CodeCheck configuration or save the configuration you are creating for later use, see *Using CodeCheck Configurations* on page 271.



Choose the checks you want to perform. The following types of options are provided:

- **SciTools' Recommended Checks:** This category lists recommended checks for your source code languages. These are standards violations that we feel are most serious.
- **Published Standards:** Collections of checks are provided to see if your code conforms to the following standards or recommendations:
 - Effective C++ (3rd Edition) Scott Meyers
 - MISRA-C 2004
 - MISRA-C++ 2008
- **Custom Checks:** Any custom checks you have installed are listed in the Checks tab. See *Writing CodeCheck Scripts* on page 272.
- **Language Specific Checks:** All checks that *Understand* can perform are listed here. They are organized first by programming language, then by category, and finally by check. Currently, most checks apply to C/C++ code, but some checks are available for other languages. You can confirm that your naming conventions are met for various entity types in most supported languages. You should check specific languages under the All Checks node; if you check the **All Checks** box, some of the checks may conflict with others, and errors are likely to occur.

- **Metrics Checks:** You can perform checks based on the values of complexity, function length, and nesting depth metrics, which are described in Chapter 9 and the list at <http://www.scitools.com/documents/metrics.php>.

Press Ctrl+F to be able to search for a check.

When you select a check, information about that check is shown in the **Detailed Description** area. You can copy the text in the description if you want to paste it into a report or email message. You can hide the description by clicking the arrowhead above the description.

For a number of checks, options are shown below the description. For example, if you select a metrics check, you can set a value that needs to be met. If you select a Naming Conventions check, you can specify a minimum and maximum length for acceptable names, any required prefix or suffix, and the types of characters and capitalization rules that names need to follow. The description provides details about the options.

Options:

Test All Const names

Minimum Length:

Maximum Length:

Required Prefix:

Required Suffix:

Character Set:

Capitalization:

Consecutive Capitals:

Check the **Use Verbose Logging** box if you want the Result Log to include a separate line for each violation found. Otherwise, the Result Log will present a summary and you can use the other tabs to sort through and view specific violations.

If you want to edit the list of checks being performed using a text file, you can do the following:

- 1 Export the entire list currently selected check (with full file paths) to a text file by clicking the  **Export Configuration to File** icon.
- 2 Edit the file with a text editor. Delete the lines for checks you do not want to apply, and save the file as plain text.
- 3 Click the  **Import Configuration from File** icon and import the file you edited. This will select only the checks listed in the file.

Once you have finished selecting the checks to perform, click **Analyze**.

Exporting the configuration to a file is also required if you want to be able to perform a CodeCheck analysis from the command line.

If you have made changes to the checks being performed, you are asked if you want to save the configuration before performing the analysis. Click **Yes** if you might want to perform this same set of checks in the future. If you were using the “New” configuration,

you are then prompted to type a name for this configuration. Type a name and click **OK**. See *Using CodeCheck Configurations* on page 271 for more information.

If more than 300,000 violations are detected, you are asked if you want to continue the check.

Viewing CodeCheck Results

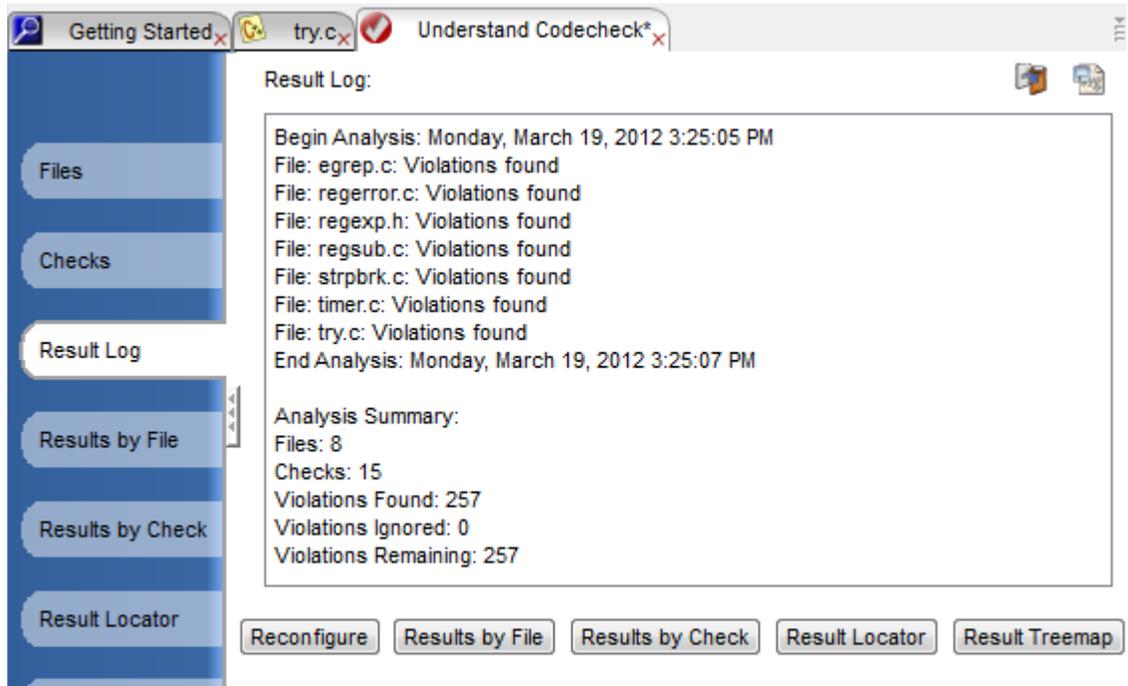
After you perform a CodeCheck analysis, you can view the results in the Results Log (page 264), Results by File tab (page 265), Results by Check tab (page 266), Result Locator tab (page 267), and Result Treemap (page 268). You can also print or export the results (page 269).

You can hide results by ignoring particular checks and violations (page 270).

Using the Result Log

When you run a CodeCheck analysis, you automatically see the **Result Log** tab, which provides a summary of the results.

The Result Log includes the number of files checked, how many checks were performed, and the number of violations found.



If you checked the **Use Verbose Logging** box in the **Checks** tab, the Result Log also includes a separate line for each violation found.

You can copy the log to your clipboard for pasting by clicking the  **Copy** icon. To save the log to a file, click the  **Export** icon.

Using the Results by File Tab

Choose the **Results by File** tab to list the problems in each file of your project.

The table lists the number of violations in each file and full file paths. Uncheck the **Show Violation Counts** box above the table to hide the number of violations. Uncheck the **Flatten Files List** box to organize files in a folder hierarchy that you can expand as needed.

Click the arrow next to a filename to expand the list of violations found in that file. The line for a violation shows the problem, the name of the entity, the line number on which the problem occurs, the number of the column (0 indexed) where the problem began, and a short description of the check performed.

Results by File: Show Violation Counts Show Ignored Violations Flatten Files List   

Results	Entity	Line	Column	Check	
4 ▲ Number of Results: 4					
1 ▶ 	C:\Program Files\SciTools\sample\fastgrep\egrep.c				
1 ▶ 	C:\Program Files\SciTools\sample\fastgrep\regerror.c				
2 ▲ 	C:\Program Files\SciTools\sample\fastgrep\timer.c				
▶ 	Violation: regerror defined but not called.	regerror	89	0	Unused Functions
▶ 	Unused Local Variable	dummy	54	13	Unused Local Var
52	<code>int ncomp, nexec, nsub;</code>				
53	<code>struct try one;</code>				
54	<code>char dummy[512];</code>				
55					
56	<code>if (argc < 4) {</code>				

Click the arrow next to a violation to see the 5 lines of code surrounding the problem. You can double-click on the code to open the source file.

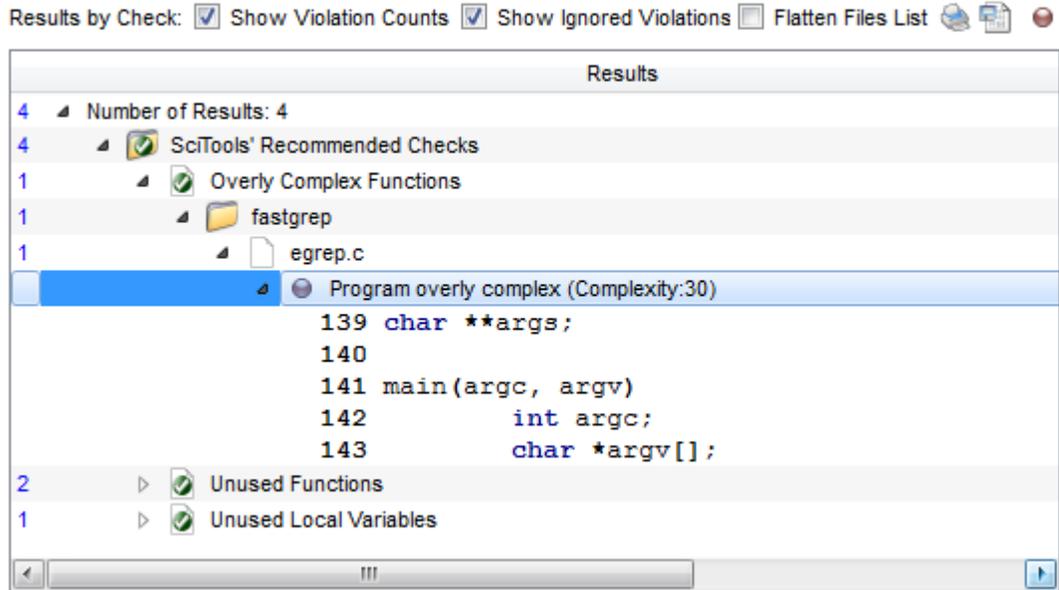
When you select a violation, the description of that check and any exceptions to the check are shown below the table. You can select text in this area and press Ctrl+C to copy it to your clipboard for pasting into other applications.

Click the  **Print** icon to print the list or the  **Export** icon to send the full list of violations to HTML files in a directory you select, your clipboard, or a text file.

Check the **Show Ignored Violations** box if you want to see the full list of violations by overriding any ignored checks and violations. See page 270 for details.

Using the Results by Check Tab

The **Results by Check** tab is similar to the Results by File tab (page 265). However, all violations of a particular type are listed together. The organization under each violation is either a list of files if the **Flatten Files List** box is checked or a folder hierarchy if the **Flatten Files List** box is unchecked.



If many violations of a particular type are detected, you might want to look at the individual checks in the **Checks** tab to see if you can set options to control the sensitivity of the checks. For example, for the “Magic Numbers” check, you can specify that bitfields can be set to fixed values and you call allow exceptions for values like 0 and 1. Another example is that for the “Functions Too Long” check, you can set the length that is considered too long and choose to ignore comment lines and blank lines.

Using the Result Locator

The **Results by Check** tab lets you search for violations using pattern matching and sorting on the file, violation name, line number, and column number.

Result Locator: Show Ignored Violations   

File	Violation	Line	Column
reg	value	≥	≥
regexp.h	Fixed Value(1) used inco...	15	21
regsub.c	Fixed Value(0) used inco...	73	35
regsub.c	Fixed Value(0) used inco...	60	29
regsub.c	Fixed Value(1) used inco...	64	30
regsub.c	Fixed Value(0) used inco...	65	25

Rationale
Using fixed values(Magic Number) in code makes it difficult to know why the engineer chose that number in choosing that number and makes it hard to adapt and extend the program. Magic numbers should be replaced with name constants to make them more usable.

Exceptions

```

13     char *regmust;          /* Internal use only. */
14     int regmlen;           /* Internal use only. */
15     char program[1];      /* Unwarranted chumminess w
16 } regexp;
```

You can type values to match filenames and violations. Right-click a column header or click the small drop-down icon to see the context menu for that column. You can choose for the filter to be case sensitive or not. You can also choose for the filter pattern matching syntax to use fixed strings (the default), wildcards, or regular expressions.

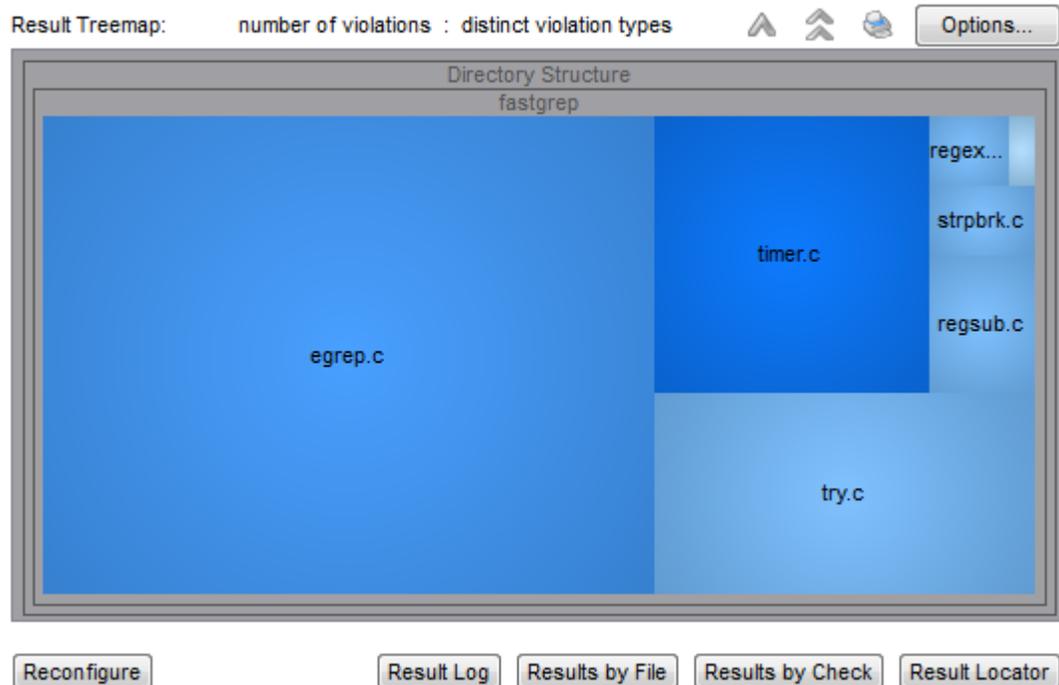
For details about using the locator fields, see *Filtering the List* on page 149.

Five lines of code surrounding the violation are shown at the bottom of the window. You can double-click this code to open the file.

Using the Result Treemap

Treemaps show metrics graphically by varying the size of node blocks and the color gradient. Each node block represents a code file. Different metrics can be tied to size and color to help you visualize aspects of the code.

CodeCheck lets you create treemaps that show the total number or density of check violations and the number of types of violations. For example, in this treemap larger block sizes indicate more violations in that file and darker blue indicates more types of violations in that file. So, while `egrep.c` has the most violations, `timer.c` has more types of violations. Notice that the text above the treemap indicates the settings used.

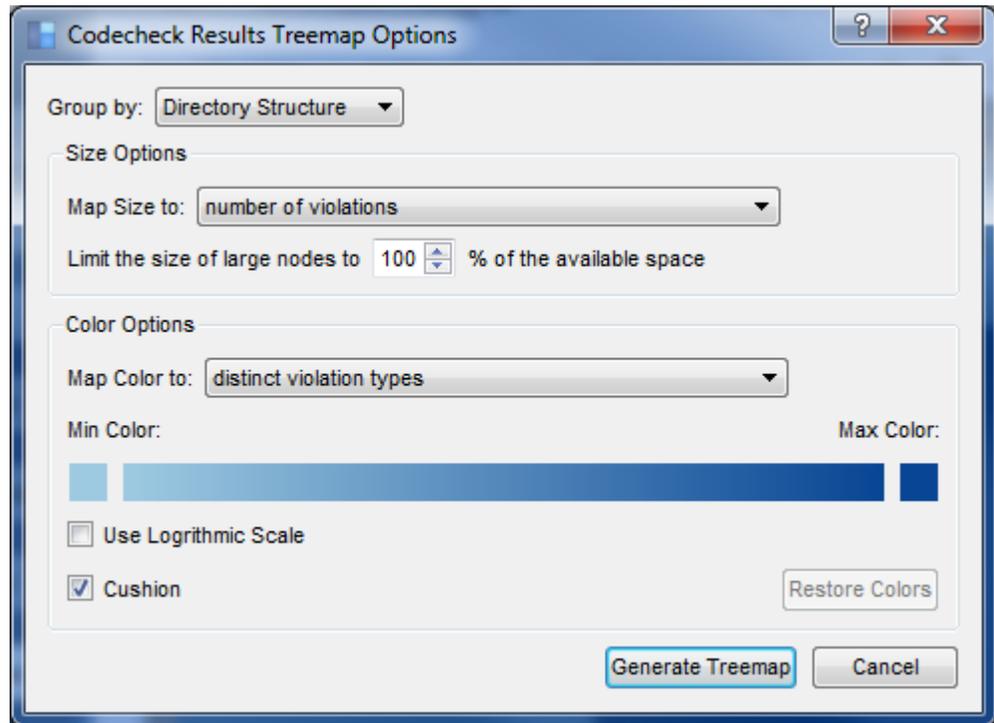


If you double-click on a file block, you see the Results by File listing with the list of violations for that file expanded.

By default, the treemap is organized using the file structure of the project as architecture nodes. Within the treemap, you can double-click on an architecture node (shown as a gray border around a set of colored blocks) to display only the contents of that node. You can also zoom in by right-clicking on a node and choosing **Drill down** from the context menu.

After drilling down in the architecture, you can use the   icons to **Pop up one level** or **Pop up all levels** the treemap. You can also right-click to use the **Pop up one level** and **Pop up all levels** commands in the context menu.

Click the **Options** button to modify which metrics are assigned to size and color:



For information about using these fields, see *Using the Metrics Treemap* on page 223.

Printing and Exporting Results

In the Result Log tab, you can copy the log to your clipboard for pasting by clicking the  **Copy** icon. To save the log to a text file, click the  **Export** icon.

In the Results by File, Results by Check, and Result Locator tabs, you can click the  Print icon to print the currently displayed results. You can click the  **Export** icon to export the detailed results to an HTML directory, your clipboard, or a text file.

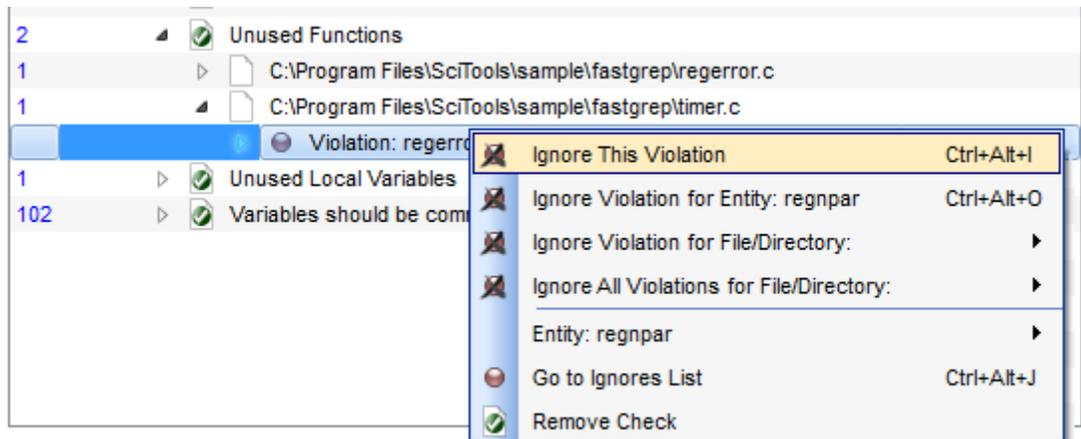
In the Result Treemap tab, you can click the  Print icon to print the currently displayed treemap diagram.

See page 257 for details about the Print Options dialog.

Ignoring Checks and Violations

A number of options let you ignore some CodeCheck violations in all or part of your project. For example, you might want to ignore violations in third-party code used by your project.

Wherever you see a violation listed in the results, you can right-click and choose to ignore the specific violation, violations of this check for the specified entity, or violations of this check for the current file or a selected directory level within the project. You can also choose to ignore all violations in a specific file or directory. If you select a directory, violations in all of its subdirectories will be ignored.



You can also click the  Ignore icon above the results to access the Ignores menu.

Violations that you choose to ignore are not listed in the Results by File or Result Locator tabs. They are highlighted with a pink background in the Results by Check tab. The violation totals in the Results by File tab *do not* include the ignored violations. Totals in the Results by Check tab *do* include ignored violations.

If you have chosen to ignore any violations, you can use the **Ignores List** tab to find and sort ignored violations. Only one item is listed if you have ignored multiple violations in a file or directory. You can search this list as you would the Result Locator list. See *Filtering the List* on page 149 for details.

To stop ignoring a violation, right-click on the item and choose **Remove from Ignores List**.

If many violations are detected, you might want to look at the individual checks in the **Checks** tab to see if you can set options to control the sensitivity of the checks. For example, for the “Magic Numbers” check, you can specify that bitfields can be set to fixed values and you can allow exceptions for values like 0 and 1.

Using CodeCheck Configurations

If you have a set of checks you want to use, you can save that list of checks as a “configuration”. Such configurations are stored outside of the project, so that you can use the same CodeCheck configuration with multiple projects.

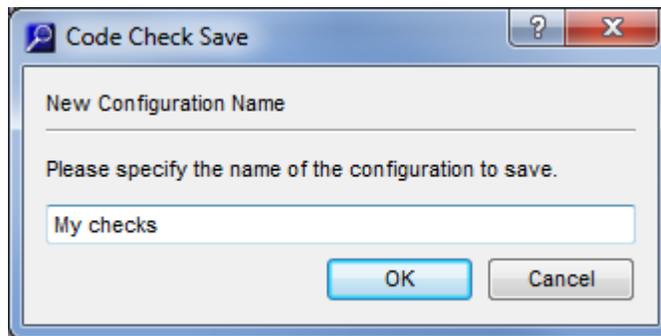
Note that the set of files to be analyzed is not saved as part of a CodeCheck configuration. The most recent set of files used is the default.

To save a configuration, follow these steps:

- 1 In the Checks tab, select the boxes for all the checks you want performed when this configuration is used.
- 2 Click the **Save** icon next to the Configuration drop-down list.



- 3 Type a name for your configuration in the dialog, and click **OK**.



You can use a configuration you have saved by selecting it in the **Configuration** drop-down list at the top of the Checks tab.

Another way to run a configuration is from the **CodeCheck > Saved Configurations** menu item, which you can use even if the CodeCheck window is not open. Or, choose **CodeCheck > Rerun Previous Checks** to rerun the most recent CodeCheck analysis.

Writing CodeCheck Scripts

CodeCheck scripts are special Perl scripts that let you provide custom checks for verifying your team's coding standards. They can be used to verify naming guidelines, metric requirements, published best practices, or any other rules or conventions that are important for your team.

You can develop these scripts using the Understand Perl API along with a set of special functions designed to interact with the Understand CodeCheck interface.

CodeCheck script files have a .upl extension.

To begin writing your own check, follow these steps:

- 1 Choose **CodeCheck > Implement Your Corporate Standard** from the menus.
- 2 In the web page this command takes you to, save the codecheck_template.upl to a file with the same name on your computer.
- 3 Edit this template file (with a text editor).
- 4 Modify the name, description, and detailed_description to match what you plan for this check to do. For example, you could use the following descriptions for a check to make sure lines do not exceed a specified length:

```
# Required - Return the short name of the check
sub name { return "Characters per line";}
```

```
# Required - Return the short description of the check
sub description { return "Lines should not exceed a set number of characters";}
```

```
# Required - Return the long description of the check
sub detailed_description { return "For readability, lines should be limited to a certain
number of characters. The default is 80 characters per line.";}
```

- 5 Modify the test_language subroutine to test for the desired languages. For example, the following test makes the check apply to C++, Java, and Python. You can look at other scripts in the \conf\plugin\SciTools\Codecheck subdirectory of your installation for more examples.

```
sub test_language {
    my $language = shift;
    return $language =~ /C++|Java|Python/;
    return 1;
}
```

- 6 If your check should be run on a per-entity basis, return 1 for the test_entity subroutine. If the check should be run only once per file, return 0 for the test_entity subroutine. For example:

```
sub test_entity { return 1;}
```

- 7 If your check should be run only once per project, return 1 for the test_global subroutine. Otherwise, return 0 for the test_global subroutine. For example:

```
sub test_global { return 0;}
```

- 8 If your check requires the user to set options, modify the `define_options` subroutine. For example:

```
sub define_options{
    my $check = shift;
    $check->option->integer("charPerLine", "Max Characters per line",80);
}
```

Modify the check subroutine to include the check and to signal a CodeCheck violation reporting the problem. The following example reports filenames that do not begin with a letter character:

```
if ($file->name =~ /^[^a-zA-Z]/){
    $check->violation(0,$file,-1,-1,"File name does not begin with a letter");
}
```

The following example reports lines longer than the specified maximum number:

```
sub check {
    my $check = shift;
    my $file = shift;
    return unless $file->kind->check("file");

    my $maxChar = $check->option->lookup("charPerLine");
    my $lineNum = 1;
    foreach my $line (split('\n',$file->contents)){
        my $length = length($line);
        if( $length > $maxChar){
            $check->violation($file,$file,$lineNum,-1,
                "$length characters on line(Max: $maxChar)");
        }
        $lineNum++;
    }
}
```

- 9 Verify that your Perl syntax is correct. The easiest way to do this is to open a command line and run the Perl application that ships with Understand: `upperl -c mysample.upl`.

To learn more, you may want to read about [Understand's Perl API](#). Browsing the CodeCheck scripts that are shipped with Understand can also be very beneficial. If you have questions about CodeCheck scripts, the [SciTools Forum](#) can be a great place to ask them.

Installing Custom Scripts

You can install one in *Understand* by dragging and dropping the script file into the *Understand* window. You will be asked if you want to install the plugin. Click **Install**.

When you install a custom check, you will see a message that identifies the directory where the check was installed. For example, `C:\Users\YourName\AppData\Roaming\SciTools\plugin\Codecheck`. You can install future checks by copying files directly to this directory.

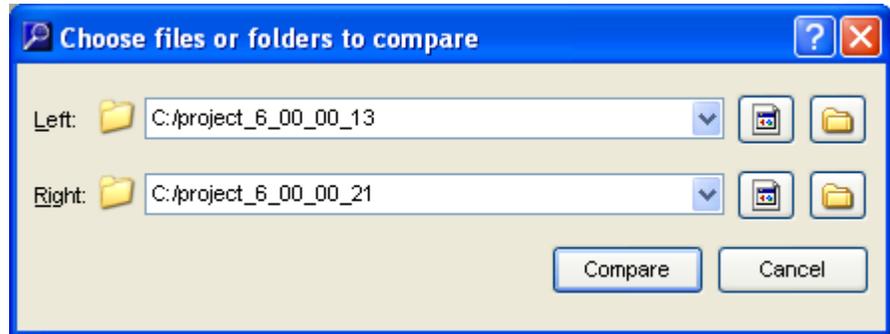
This chapter explains the source-code comparison features provided by *Understand*.

This chapter contains the following sections:

Section	Page
Comparing Files and Folders	275
Comparing Entities	277
Comparing Text	278
Exploring Differences	279

Comparing Files and Folders

Understand provides a tool for comparing files and folders. To open this tool, choose **Tools > Compare > Compare Files/Folders** from the menus.



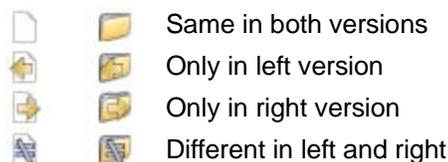
In this dialog, select a file or folder for the left and right comparison. Both sides should be similar files or similar folders. Click the file button to browse for a file; click the folder button to browse for a directory.

Subdirectories of the directories you choose are also compared.

When you click **Compare**, the comparison begins. The status bar at the bottom of the *Understand* window shows what is being compared.

The lower sections of the comparison results window are described on page 279. If you are comparing folders, there is an additional top section that lets you see the folder-level and file-level differences and select individual files whose contents you want to compare.

The comparison uses the following folder and file icons.



By default, all files and folders are listed. You can use the **Show** drop-down to choose whether to restrict the list to showing only:

- **Different:** Show files and folders that either exist in only one version or are different in the two versions.
- **Left Only:** Show files that are contained in the left version only. All different folders are shown because some may contain files that are only in the left version.
- **Right Only:** Show files that are contained in the right version only. All different folders are shown because some may contain files that are only in the right version.
- **Same:** Show files that are the same in both versions. All folders are shown because some that are different may contain files that are the same.



The **Filter** field lets you type characters you want to match in the directory path or filename. For example, “sim” matches any folders or files with “sim” in their names. All files within folder that match the **Filter** (and the **Show** drop-down setting) are shown. Filtering occurs as you type. Wildcards and regular expressions are not recognized.

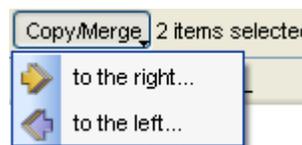
You can change the colors use for folder and file names by choosing a color from the **Colors** menu and selecting a new color in the color picker.



You can highlight all items that exist in only the left or right version. To do this, first right-click on the file list and choose **Expand All**. Then click the **Select** button and choose either **Orphans left** or **Orphans right**. You will see a warning that some items may have been skipped; this applies only if you did not use **Expand All**.



You can copy folders and files from one side to the other. The copied items overwrite any items with the same names. To copy, first select the items you want to copy. (To copy a folder and its contents, select the folder *and* all the folders and files it contains.) Then click the **Copy/Merge** button and choose either **to the right** or **to the left**. This opens the Copy Files dialog, which lists the files or folders to be copied. If the list is correct, click **OK**.



You can save changes you make to files in the file and folder comparison. If you have modified a file on the right, you can click the  **Save** icon to save that file to its existing location. You can use the  **Save As** icon on either the left or right to save a file to a different location.

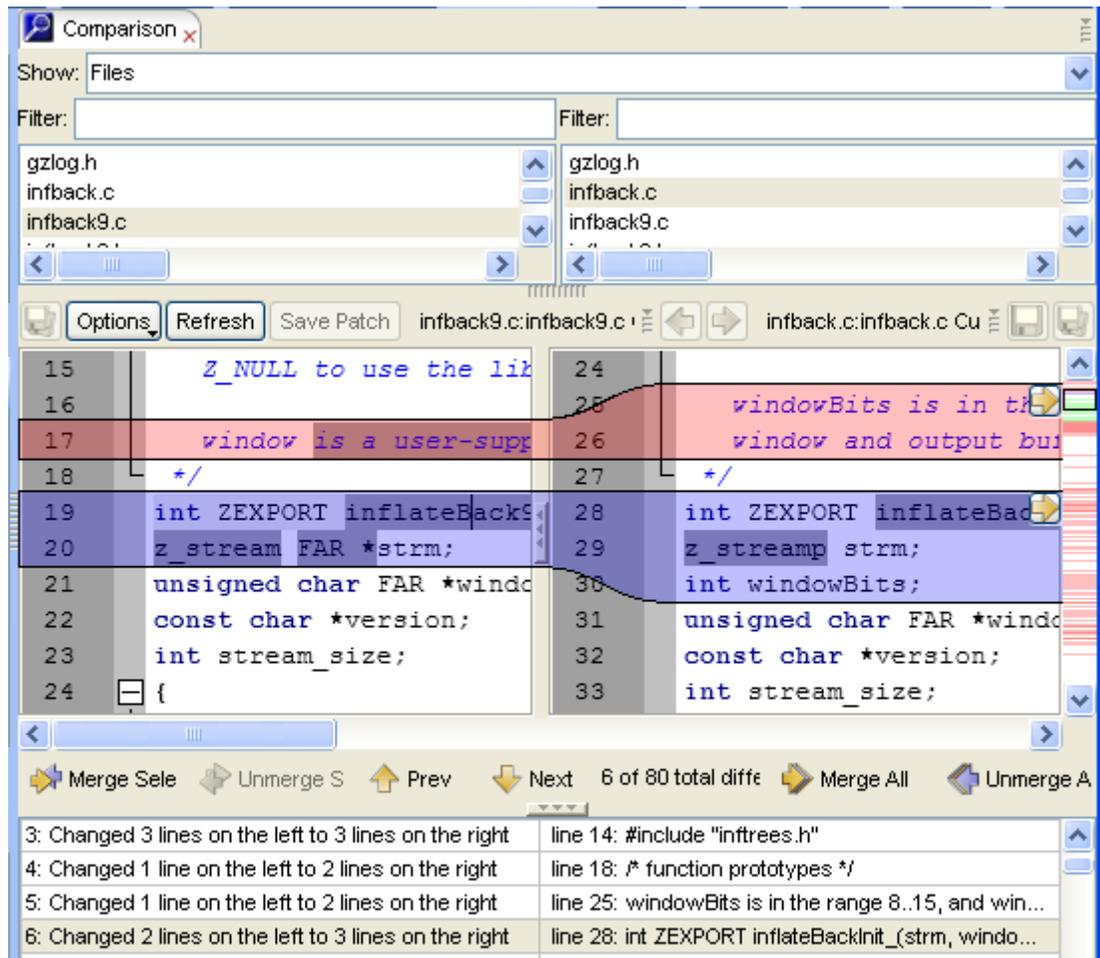
If you have modified a file in the comparison area and then use the folder and file list to switch to another file, you are asked whether you want to save the file.

By default, the file on the left is in read-only mode, and the file on the right is in read-write mode. You can change the mode for either file by clicking in the file and then clicking “RO” or “RW” in the status bar to toggle the mode.



Comparing Entities

You can compare two entities by choosing **Tools > Compare > Compare Entities** from the menus. You see the Comparison window.



The middle and lower parts of this comparison behave similarly to other comparisons. At the top of the comparison is an entity filter (page 123). Select a type of entity in the **Show** drop-down. Then use the lists to select two entities you want to compare.

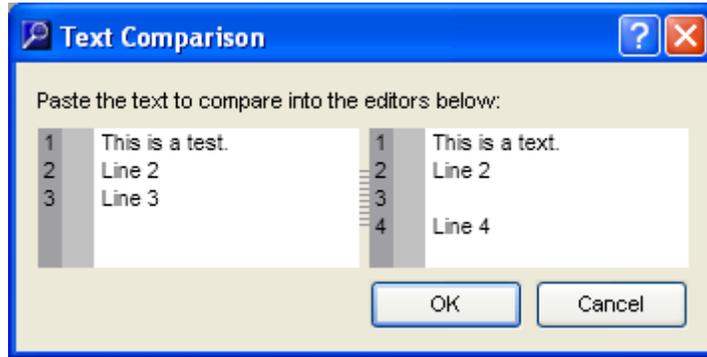
The **Filter** fields let you type characters you want to match anywhere in the entity name. Filtering occurs as you type. Wildcards and regular expressions are not recognized.

If you are comparing an entity other than a file (such as a function), merging changes and saving files in the comparison is not permitted. You can still use the **Save Patch** button to create a patch file in “unified format”.

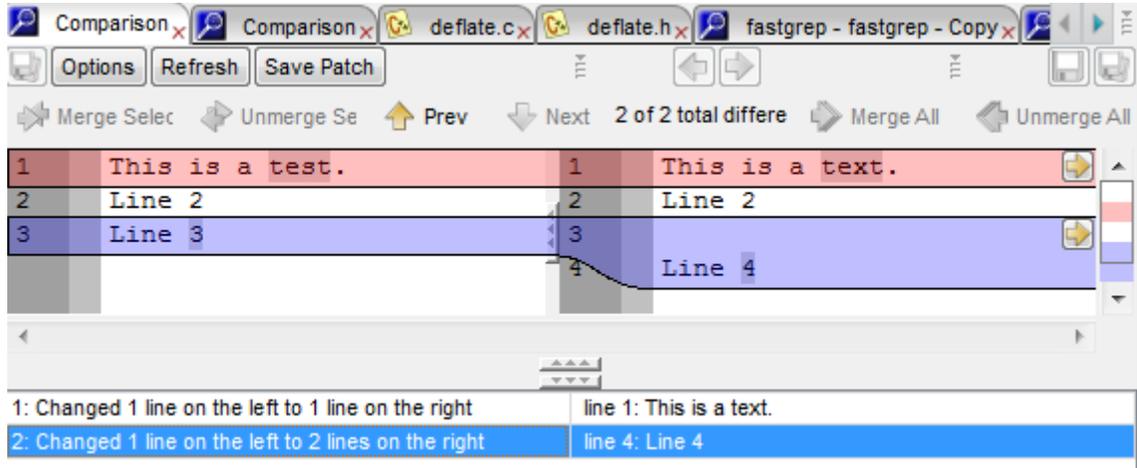
If you are comparing a file, you can merge changes and use the **Save** and **Save As** buttons for the version of the file on the right.

Comparing Text

You can compare text that you paste into a window by choosing **Tools > Compare > Compare Arbitrary Text** from the menus. You see a window like this:



Paste the before and after text you want to compare into the left and right sides. Then, click **OK** to see the comparison.



The text comparison is similar to the comparison between two entities. You can merge and unmerge differences, but cannot save files.

Click the  fold icon to hide or view the patch file syntax and/or the list of differences.

Exploring Differences

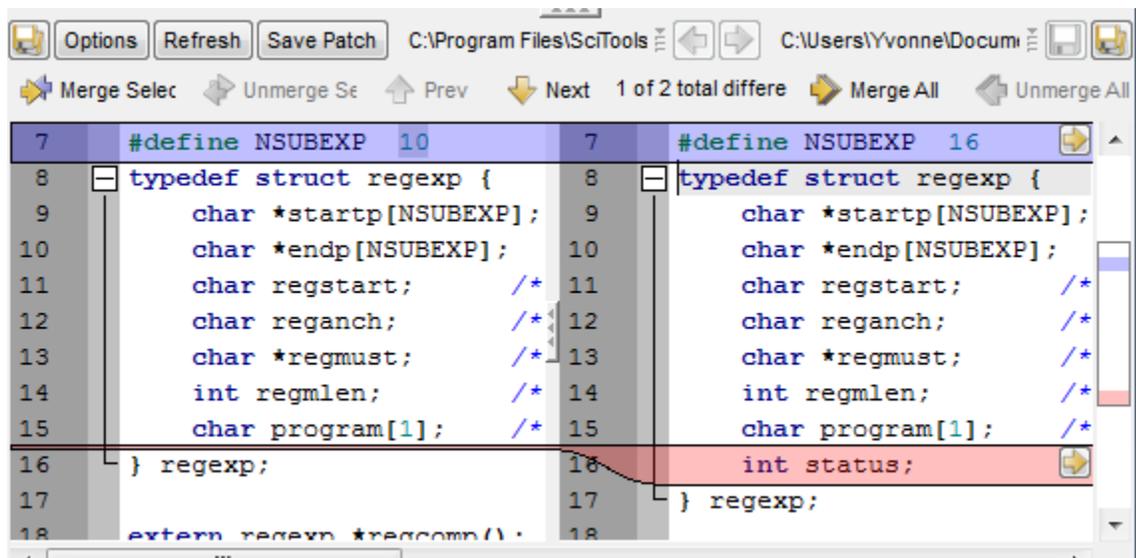
When you compare items, you see a comparison window. Depending on what you are comparing, you see several of the following areas that help you navigate the differences:

- **Changed Entities:** This area lets you select files or entities to compare. It differs depending on what you are comparing.
- **Code Comparison:** This area allows you to examine the differences in the code. See page 279.
- **Patch File:** The patch area shows the patch file syntax to convert from the left version to the right version. See page 282.
- **Difference List:** This list allows you to select individual differences between two versions. See page 282.

The small  fold icon between the areas allows you to close and reopen areas to make more space for the other areas. If you point your mouse to the right or left of either fold icon, you see the pane resize mouse cursor, which allows you to resize the areas as needed. 

Code Comparison

The Code Comparison area shows individual differences between versions of an entity. The display is similar to that of common differencing tools.

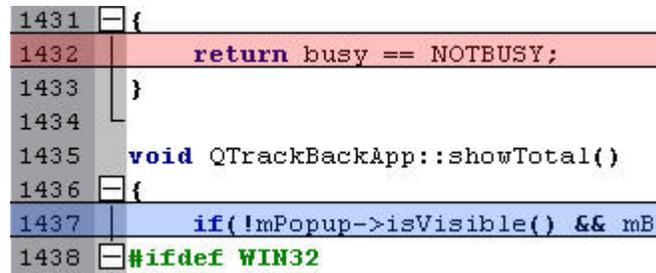


The left side shows the code from the first item you are comparing; the right side shows the code from the second item. The entity path is shown just above the code.

Scrolling of the two versions is synchronized horizontally and vertically. The scrollbar shows the location and size of changed sections of code using the comparison colors.

For certain languages that *Understand* understands—such as C code—you can click the + and - signs in the code to expand and compress code constructs such as if and else statements, functions, extended comments, and so on.

The currently selected difference is highlighted in blue (or bluish purple on some screens) by default. Other differences are highlighted in pink by default.



```

1431  {
1432  return busy == NOTBUSY;
1433  }
1434
1435  void QTrackBackApp::showTotal()
1436  {
1437  if(!mPopup->isVisible() && mB
1438  #ifdef WIN32

```

You can use the small fold icon (like the one shown here) between the two code versions to hide the left version of the code temporarily. Or, click the  right arrow next to a code change to do the same thing.

You can edit the source code if you like in the right version of the files. You cannot save code directly to a file. Instead, you can use the  button to save a patch file or you can copy and paste code with merged differences and edits into another application.

You can select text and copy it to the clipboard. To select text, use the mouse or your keyboard. To select all, press Ctrl+A or right-click and choose **Select All**. To copy text to the clipboard, press Ctrl+C or right-click and choose **Copy**.

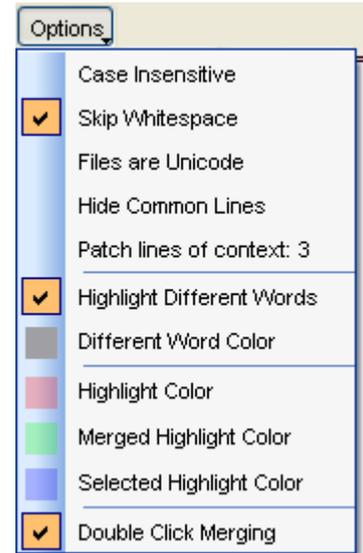
As always, right-click on any entity name or other text in the code to see lots of item-specific options in the context menu.

The status bar at the bottom of the window shows your line location in the source code where you last clicked.

The toolbar at the top of the Code Comparison area contains the following controls:

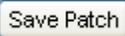
 **Options button:** Use the Options drop-down to set the following options:

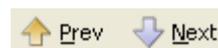
- **Options->Case Insensitive:** By default, changing the case of a letter is not treated as a difference. For example, if you change “a” to “A”, the Difference List shows “No Differences” if that was the only change.
- **Options->Skip Whitespace:** By default, changing the number of spaces or tabs is not treated as a difference. The Difference List shows “No Differences” if only whitespace was changed. You can change this behavior by toggling this option off.
- **Options->Skip Blank Lines:** By default, a different number of blank lines is treated as a difference. You can change this behavior by toggling this option on.
- **Options->Files are Unicode:** By default, differences are reported only for ASCII files. If *Understand* says “File is Binary”, use this command to turn on Unicode file handling.
- **Options->Hide Common Lines:** By default, all lines in both files are shown. If you check this option, most lines that are the same in both versions are hidden in the left (older) version.
- **Options->Patch lines of context:** The patch area shows the patch file syntax to convert from the left version to the right version. By default, 3 matching lines are shown around a change to provide context. You can choose this option and change the number of lines in the Patch Lines of Context dialog.
- **Color choices:** These options let you change the highlighting in the code comparisons. The Different Word color is an overlay that is combined with the other highlight colors as appropriate.
- **Options->Double Click Merging:** A shortcut for merging is to double-click on a difference in the code. This works only if you enable it here.



The Case Insensitive, Skip Whitespace, and Files are Unicode options are not available if you have made a change to a file.

 **Refresh button:** You can use the Refresh button to update the Difference List at the bottom of the Change Results. This list may become out-of-date if you merge differences or edit the file directly.

 **Save Patch button:** You can use the Save Patch button to create a patch file in “unified format” (or unidiff). This patch file can be used with the Unix patch tool and other similar programs.



Click the **Prev** and **Next** buttons above the Code Comparison area to jump to another difference between the entities.



You can merge differences into the version of an entity shown on the right. You cannot save code directly to a file. Instead, you can save a patch file or copy and paste code with merged differences and edits into another application. To merge differences, follow these steps:

- 1 Select a difference in the code or by selecting a line in the Difference List area.
- 2 Click the **Merge Selected** button. This copies the older (left) version of this difference to the current (right) version of the code. (If you change your mind, click **Unmerge Selected**.)
- 3 Click the **Prev** or **Next** button to move to another difference and repeat the previous step.

In the Difference List, merged differences are shown in blue italics. In the code, differences you have merged are highlighted in green. (The currently selected difference is still highlighted in blue/purple, even if it has been merged.)

A shortcut for merging is to double-click on a difference in the code if you have enabled **Double Click Merging** in the Options drop-down.



If you know you want to merge all of the differences, click **Merge All**. If you want to undo all merges you have made, click **Unmerge All**.

Patch File

This area shows the differences in patch file format. Such patch files can be used with the Unix patch tool and other similar programs. You can hide this area by clicking the small  fold icon above the area.

The **Patch lines of context** command in the **Options** button menu lets you adjust the number of unchanged lines shown around a difference.

Difference List

The Difference List area shows a list of the differences in the code shown in the Code Comparison area.

1: Changed 1 line on the left to 2 lines on the right	line 114: s->stream.transparent = (voidpf)0;
2: Deleted 1 line on the right	line 117: s->file = NULL;
3: Changed 1 line on the left to 1 line on the right	line 123: s->crc = crc64(0L, Z_NULL, 0);

In the Difference List, merged differences are shown in blue italics.

You can hide the Difference List portion of the results by clicking the small  fold icon below the area. This makes more space for the Code Comparison area.

Running Tools and External Commands

This chapter will show you how to configure and use source code editors and other external tools from within *Understand*.

This chapter contains the following sections:

Section	Page
Configuring Tools	284
Adding Tools to the Context Menus	291
Adding Tools to the Tools Menu	292
Adding Tools to the Toolbar	293
Importing and Exporting Tool Commands	294
Running External Commands	295

Configuring Tools

Select **Tools > User Tools > Configure** from the menus to open the Tool Configurations dialog, where you can configure external tools such as source code editors for use within *Understand*. External tools configured for use will be available for context-sensitive launching. The Tool Configurations dialog provides a number of categories that determine how they are launched.

The screenshot shows the 'User Tools' configuration dialog. At the top, there is a list of tools: 'Edit Entity with TextPad', 'Edit Location with TextPad' (which is selected), and 'New User Tool'. To the right of this list are three buttons: 'New', 'Delete', and 'Import'. Below the list, there are several input fields and dropdown menus: 'Menu Text' (containing 'Edit Location with TextPad'), 'Command' (containing 'txtpad32.exe'), 'Parameters' (containing '\$CurEntity'), 'Initial Directory', 'Icon file', 'Input' (set to 'None'), and 'Output' (set to 'Capture'). There are also checkboxes for 'Understand perl script', 'Analysis Options' (with sub-options 'Save All', 'Rescan', 'Analyze Changed Files', 'Analyze All Files'), and 'Run Analysis Options' (with sub-options 'Before User Tool', 'After User Tool'). At the bottom, there is an 'Add to...' section with checkboxes for 'Pop Up Menu', 'Main Menu', and 'Toolbar', all of which are checked. Finally, there are 'OK' and 'Cancel' buttons at the bottom right.

First, use the **User Tools** category of the Tool Configurations dialog to define a command and parameters as follows:

- 1 Click **New**.
- 2 In the **Menu Text** field, type the name you want to appear in *Understand* menus for this tool. You can use variables in the Menu Text. For example, you can use `$CurEntity` to put the name of the currently selected entity in the tool name. See *Variables* on page 286 for a full list of variables.

- 3 If the tool you use is on your executable search path, simply type its name in the **Command** field. If not, use the **Browse** button to specify the full path to its executable.
- 4 In the **Parameters** field, specify parameters that need to be passed on the tool's command line. See *Variables* on page 286 for a full list of variables. Variables beginning with \$Cur are current position variables that apply only from a Source Editor window. Variables beginning with \$Decl are declaration variables that apply only when an entity with a declaration is selected. Variables beginning with \$Prompt display a dialog to ask the user for some information.
- 5 In the **Initial Directory** field, specify the directory in which the tool should start running. You can use variables such as \$CurProjectDir in this field.
- 6 In the **Icon file** field, type or browse for a small graphic file to act as the icon for this command. You can choose a BMP, GIF, PBM, PGM, PNG, PPM, XBM, or XPM file.
- 7 Choose the **Input** you want to use for the command. The options are **None** (default), **Selected Text**, and **Entire Document**. The Selected Text and Entire Document options are intended to be used when running a tool from the Source Editor.
- 8 Choose what you want done with the **Output** from the command. Options are:
 - **Discard** the output. This is the default.
 - **Capture** it in a Command Window, which is an area that appears by default near the Information Browser. The command window is reused by default if you run another tool or re-run the same tool. You can force results to go to a new window by unchecking the Reuse box on the command results window(s).
 - **Replace Selected Text** in the current Source Editor window.
 - **Replace Entire Document** in the current Source Editor window.
 - **Create a New Document** in a Source Editor window.
 - **Copy to Clipboard** so you can paste the results elsewhere.
- 9 Check the **Understand perl script** box if this is a Perl script that uses the Understand Perl API.
- 10 In the "Analysis Options" area, choose actions you would like to be performed before and/or after this user tool is run. These actions can include saving all files, re-scanning for new files in project directories, analyzing modified files, and analyzing all files.
- 11 In the "Add to..." area, choose ways you want to access this command in *Understand*. The **Pop Up Menu** checkbox adds the tool to the right-click context menu. The **Main Menu** checkbox adds the tool to the **Tools > User Tools** submenu. The **Toolbar** checkbox adds the tool's icon to the toolbar.

To edit settings for an existing tool, select it in the list and make changes as needed. Click **OK** to save your changes. If you want to remove a tool, select it and click **Delete**.

For information about using the **Import** button, see *Importing and Exporting Tool Commands* on page 294.

Variables

Variables beginning with \$Cur are current position variables that apply only from a Source Editor window. Variables beginning with \$Decl are declaration variables that apply only when an entity with a declaration is selected. Variables beginning with \$Prompt display a dialog to ask the user for some information.

You can use the following variables in the Command or the Parameter field.

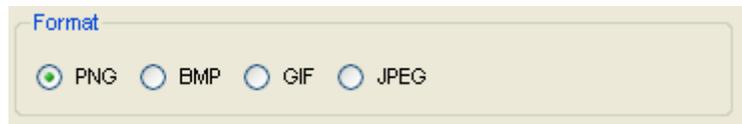
Variable	Description
\$CplusplusIncludes	Lists all of the include directories specified in the Project Configuration. This may be useful, for example, if the tool you want to run is a compiler or linker.
\$CplusplusMacros	Lists all of the macro definitions specified in the Project Configuration.
\$CurCol	Column position of cursor position in current file.
\$CurEntity	Full name of selected entity.
\$CurEntityShortName	Short name of selected entity.
\$CurEntityType	Type of selected entity.
\$CurFile	Current file's full path.
\$CurFileDir	Current file's directory.
\$CurFileExt	Current file's extension.
\$CurFileFlatStr	Current file's full path with all directory separation characters (such as / and \) replaced with an underscore (_).
\$CurFileName	Current file's name not including extension or full path.
\$CurFileShortName	Current file's name without full path.
\$CurLine	Line number of cursor position in current file.
\$CurProject	Current fullname location of opened project.
\$CurProjectDir	Directory in which the opened project is located.
\$CurProjectName	Current short filename of opened project (not including extension).
\$CurReportHtml	Current fullname location of opened project's HTML report.
\$CurReportText	Current fullname location of opened project's CSV report.
\$CurScopeEntity	Scope of current entity.
\$CurSelection	Currently selected text in the current window (file windows only).
\$CurWord	The word/text at the current cursor position in the current file window.
\$DeclCol	Column in which the selected entity was declared, defaults to 1.
\$DeclFile	Full path name of the file in which the selected entity was declared.
\$DeclFileShortName	Filename only of the file in which the selected entity was declared.
\$DeclLine	Line in which the selected entity was declared, defaults to 1.
\$DeclScopeEntity	Name of the entity within which the selected entity is declared.

Variable	Description
\$NamedRoot	Specify \$NamedRoot "namedrootname", where the namedrootname is the actual name of the named root. Note that the named root must be active. This variable can be used in either the Parameters field or the Initial Directory field.
\$PromptForCheckBox	Prompts user for a true/false value required by the command. A 0 (unchecked) or 1 (checked) is passed to the command in place of this variable. This variable should be followed by a string to be displayed as text next to the checkbox. For example, <code>\$PromptForCheckBox "Show Debug Text"</code> displays the following prompt <div data-bbox="954 636 1211 680" data-label="Image"> </div>
\$PromptForCheckBoxGH	Prompts user with a series of checkboxes displayed in a horizontal group. For example, <code>\$PromptForCheckBoxGH "Show=Debug Text;Tool Tips;Line Numbers"</code> displays the following prompt. The label ("Show" in this example) is optional. A semicolon must be used to separate items. The text strings for all checked items (separated by spaces) are passed to the command. <div data-bbox="716 917 1446 1045" data-label="Image"> </div>
\$PromptForCheckBoxGV	Prompts user with a series of checkboxes displayed in a vertical group. For example, <code>\$PromptForCheckBoxGV "Show=Debug Text;Tool Tips;Line Numbers"</code> displays the following prompt. The text strings for all checked items (separated by spaces) are passed to the command. <div data-bbox="721 1255 1458 1476" data-label="Image"> </div>
\$PromptForDir	Prompts user to select a directory and passes the full path as a string. For example, <code>\$PromptForDir "Directory Path=\$CurProjectDir"</code> displays the following prompt with the current project directory as the default. The "..." button opens the standard directory selection dialog for your operating system: <div data-bbox="711 1692 1442 1738" data-label="Image"> </div>

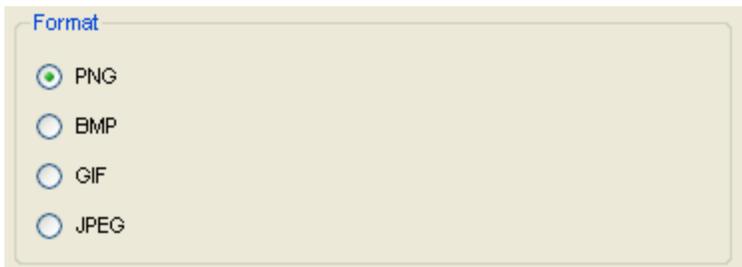
Variable	Description
\$PromptForFile	Prompts user to select a file and passes the full path as a string. For example, <code>\$PromptForFile "Filename=\$CurFile"</code> displays the following prompt with the current source file as the default. The “...” button opens the standard file selection dialog for your operating system:



\$PromptForRadioBoxGH	Prompts user for a selection from a set of options displayed horizontally. For example, <code>\$PromptForRadioBoxGH "Format=PNG;BMP;GIF;JPEG"</code> displays the following prompt. The text string for the selected item is passed to the command.
-----------------------	---



\$PromptForRadioBoxGV	Prompts user for a selection from a set of options displayed vertically. For example, <code>\$PromptForRadioBoxGV "Format=PNG;BMP;GIF;JPEG"</code> displays the following prompt. The text string for the selected item is passed to the command.
-----------------------	---



\$PromptForSelect	Prompts user to select from a drop-down box. For example, <code>\$PromptForSelect "Build Version=Debug;Release;Optimized"</code> displays the following prompt. The text string for the selected item is passed to the command.
-------------------	---



Variable	Description
\$PromptForSelectEdit	Prompts user to select from a drop-down box or edit the text in the box. For example, <code>\$PromptForSelectEdit "Build Version=Debug;Release;Optimized"</code> displays the same prompt as the example for <code>\$PromptForSelect</code> , except that you can edit the string in the box.
\$PromptForText	Prompts user for a string required by the command. For example, <code>\$PromptForText "Replace=foo"</code> displays the following prompt and provides a default value. The text provided is passed as a string.



In general, the multiple-selection `$Prompt` variables accept strings of the format `"label=item1;item2"`. Any number of items may be separated by semicolons. The item strings for all selected items (separated by spaces) are passed to the command.

The label is optional except in the cases of `$PromptForCheckBox`, `$PromptForDir`, `$PromptForFile`, and `$PromptForText`. The default value is optional in the cases of `$PromptForDir`, `$PromptForFile`, and `$PromptForText`.

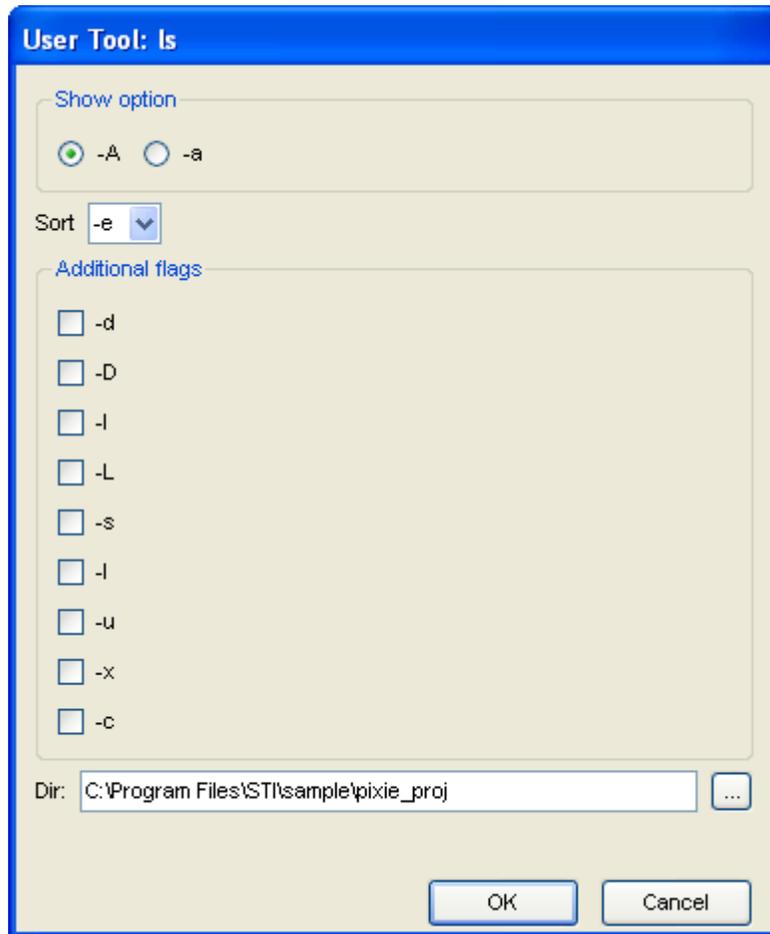
Prompts are processed after the other types of variables, so you can use other variables in the labels and values. For examples, see `$PromptForDir` and `$PromptForFile` in the previous table.

In addition, operating system environment variables can be used in prompt syntax. For example, `$PromptForSelect "Dir=$PATH"` presents a drop-down list of all the directory paths in your `$PATH` definition.

You can optionally provide the item list in a separate file. In that case, the syntax for most `$Prompt` variables is `label=@fullpath_of_listfile.txt`.

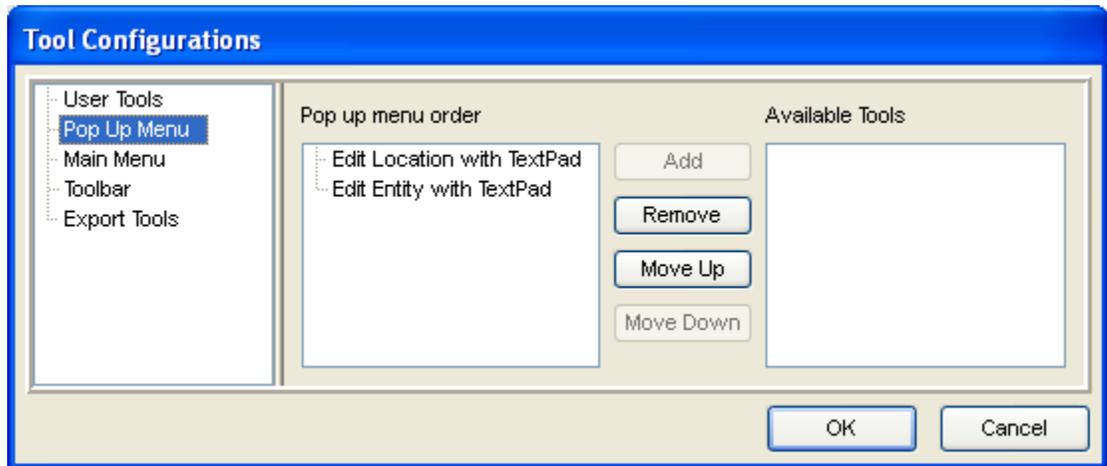
You can combine variables to pass all the parameters needed by a command. All prompts are combined into one dialog. For example if the command is "ls", you can use the following parameters to create a dialog that lets you select command-line options for the ls command:

```
$PromptForRadioButtonGH "Show option=-A;-a" $PromptForSelect "Sort=-e;-t" $PromptForCheckBoxGV "Additional flags=-d;-D;-l;-L;-s;-l;-u;-x;-c" $PromptForDir "Dir:=$CurProjectDir"
```



Adding Tools to the Context Menu

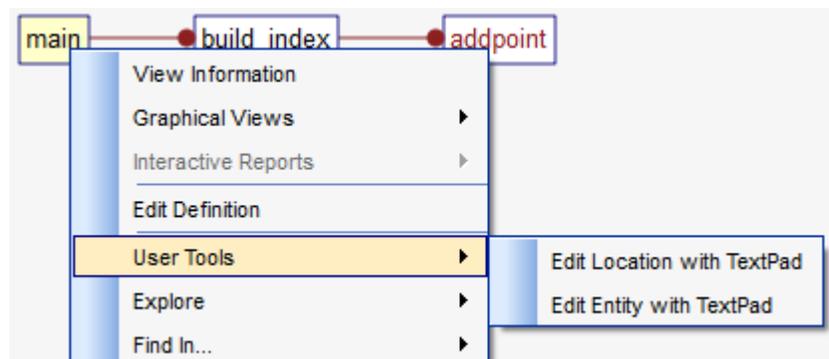
Once a command is defined in the Tools tab, the **Pop Up Menu** category in the Tool Configurations dialog lists user tools that are currently in the context menu on the left and commands you can add to that menu on the right. (Context menus are sometimes called contextual, shortcut, right-click, or pop-up menus.)



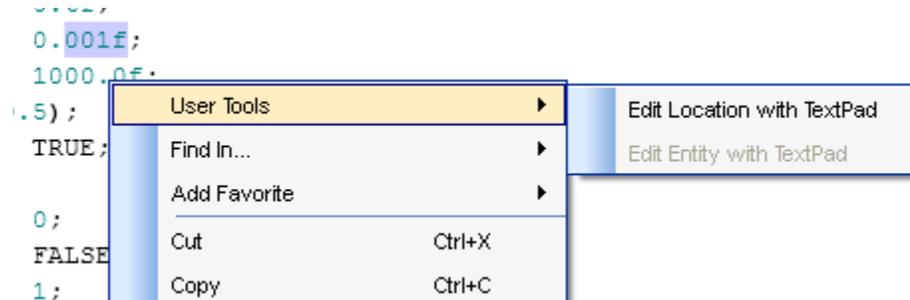
To add a tool to the context menus, select it on the right and click **Add**. To remove a tool from the context menus, select it on the left and click **Remove**.

User tools appear in the context menu in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the tools as desired.

The following figure shows a context menu for an entity showing the available external tools.

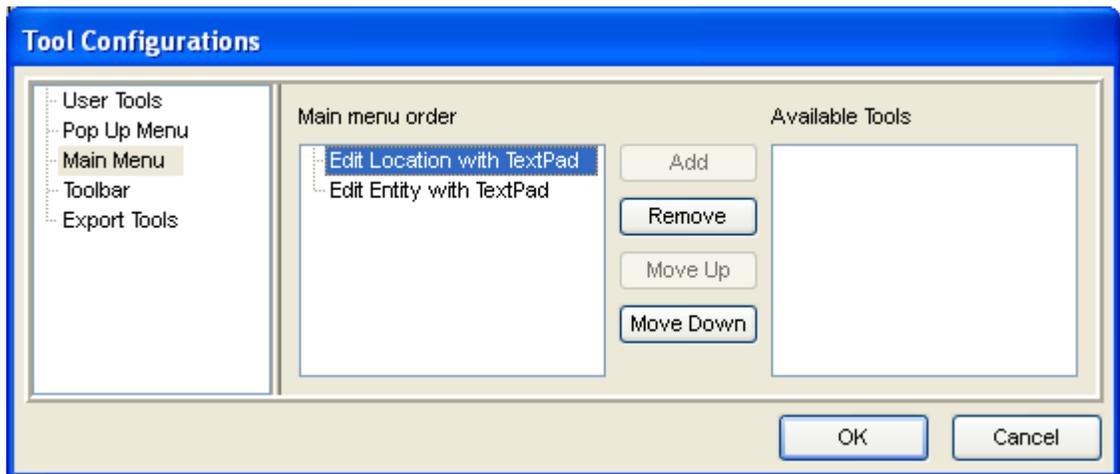


Tools are active or inactive on the context menu based on the context of the parameters provided to the tool. For example, a source editor that specifies \$DeclFile as a parameter is selectable from the context menu for any entity where the declaration is known, but will not be active for an undeclared entity or when no entity is selected.



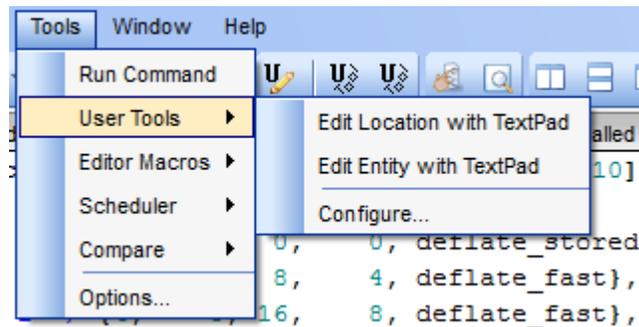
Adding Tools to the Tools Menu

Once a command is defined in the Tools tab, the **Main Menu** category in the Tool Configurations dialog lists user tools that are currently in the **Tools > User Tools** menu on the left and commands you can add to that menu on the right.



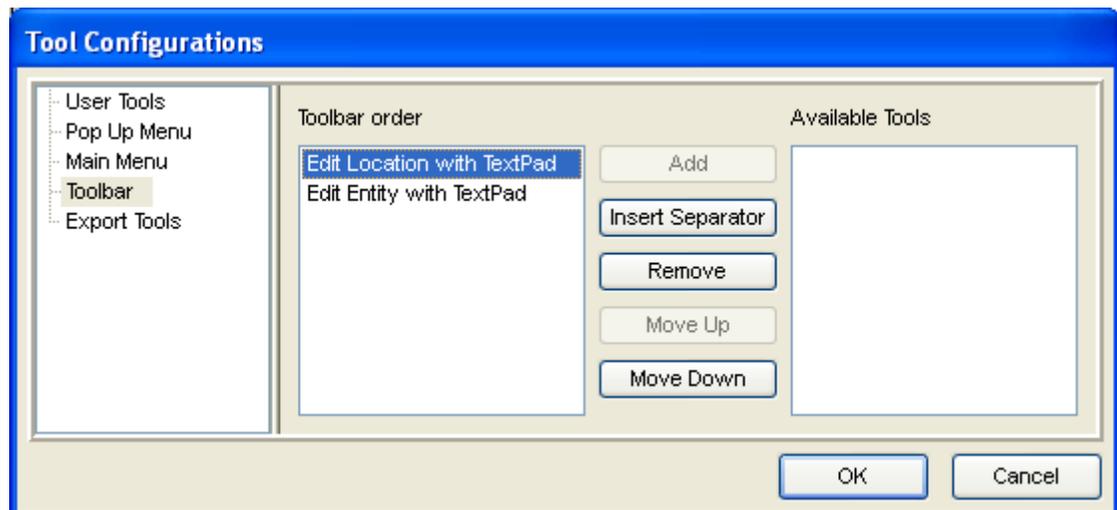
To add a tool to the menus, select it on the right and click **Add**. To remove it from the menus, select it on the left and click **Remove**.

User tools appear on the **Tools** menu in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the tools as desired.



Adding Tools to the Toolbar

Once a command is defined in the Tools tab, the **Toolbar** category in the Tool Configurations dialog shows user tools currently in the toolbar in the left box and commands you can add to the toolbar in the right box.



To add a tool to the toolbar, select it on the right and click **Add**. To remove it from the toolbar, select it on the left and click **Remove**.

To add a vertical separator to the toolbar, select the item in the Toolbar order box that should have a vertical line to the right of it. Click **Insert Separator** to add "-----" to the list.

Icons for the selected tools appear on the toolbar in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the icons as desired.

To change the icon for a particular tool, use the **Icon file** field in the User Tools category.

For example, in the following figure, the first icon is provided by *Understand* to open the Tool Configurations dialog. The second icon is the default icon for a user tool if none is specified.



In this toolbar, two icons have been added for user tools. A separator has been added between them.



Note: You can control which icons are visible in the main toolbar by right-clicking on the background of the toolbar and checking or unchecking items for the various toolbar sections.

Importing and Exporting Tool Commands

You can import and export tool commands from files. This makes it easy to share tool commands with co-workers.

- 1 To export commands, choose **Tools > User Tools > Configure** from the menus and switch to the Export Tools category. You will see the following dialog.



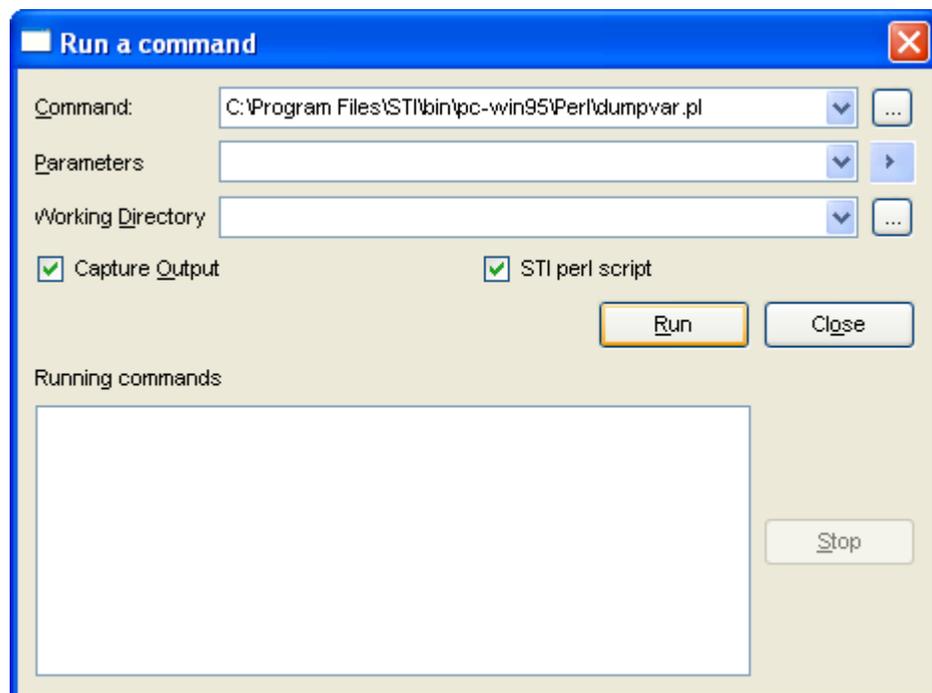
- 2 Check the boxes next to commands you want to share.
- 3 Click **Export to file**.
- 4 Choose a location and filename for an initialization file (*.ini) that contains the selected user tool information.
- 5 Click **Save**.

To import commands, choose the User Tools category in the Tool Configurations dialog and click the **Import** button. Browse for an initialization file created by another *Understand* user and click **Open**. In the Import User Tools dialog, check the boxes next to the tool commands you want to be available in your copy of *Understand*.

Running External Commands

The **Tools > Run Command** menu item permits any external command to be run directly from *Understand*. Common commands to invoke are compilers, configuration management tools, and Perl programs written using *Understand*'s API.

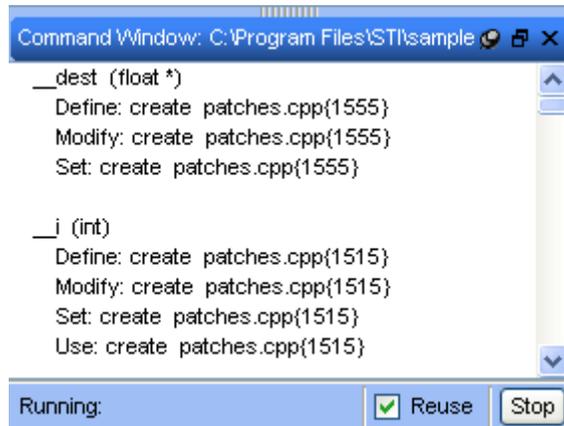
The **Run a Command** dialog looks like this:



To run a command, follow these steps:

- 1 Type a **Command** or click ... and browse for a file to run. A number of Perl programs are provided in the *Understand* installation.
- 2 Type any command-line **Parameters** required by the command. Click the right arrow if you want to select one of the special variables. These are listed on page 286.
- 3 Click ... and browse for the directory that should act as the **Working Directory**.
- 4 If you want the output sent to a window in *Understand*, leave the **Capture Output** box checked.
- 5 If you are running a Perl script, check the **STI Perl script** box if this is a script provided by Scientific Toolworks.
- 6 Click **Run**. The output is shown in a Command Window in *Understand* if you checked the **Capture Output** box. Otherwise, the command runs in the background

and output is shown in the **Running Commands** box. You can select a command from this list and click **Stop** to halt the command.



The font used in the Command Window is determined by settings in the Command Window category of the Understand Options dialog, which you can open by choosing **Tools > Options** from the menus. See page 102.

On UNIX systems, output to both stdout and stderr are captured.

This chapter shows how to use *Understand* from the command line. Command line processing can be used in a batch file for automatic re-building and report generation of projects.

This chapter describes the “und” command line, which allows you to analyze sources and create *Understand* databases from the command line. In addition, it allows you to generate metrics and reports.

Note: The “und” commands were standardized in build 571, and the tool should now be much easier to use. Because of the extensive changes, this new version is not backwards compatible with older versions of und. The old und executable has been renamed “undlegacy”. If you have legacy scripts, you should rename the binary run by these scripts in order for them to continue to work.

Most examples in this chapter refer to C/C++ files. However, you can use “und” with any supported language.

This chapter contains the following sections:

Section	Page
Using the und Command Line	298
Using the understand Command Line	305
Using Buildspy to Build Understand Projects	306

Using the und Command Line

The command-line tool for creating and building *Understand* databases is *und*.

The *Understand* installer can optionally place the appropriate bin directory in your operating system's PATH definition to simplify running the "und" command line. For example the Windows PATH definition might include the C:\Program Files\SciTools\bin\pc-win64 path.

Und can be run in the following modes:

- **Interactive mode:** You enter this mode if you simply type `und` on the command line with no command or text file. While in the interactive shell, settings such as open database are remembered from command to command. This is a good mode to use to test a sequence of commands you want to use in a batch file. You can optionally specify the database to open on the command line to run the interactive shell.

```
c:\Program Files\SciTools\bin\pc-win64>und
Welcome to und. Type "help" for a list of commands. "quit" to quit
und> _
```

- **Batch mode:** Once you identify a sequence of commands you want to run more than once, you can store them in a text file that you can run in batch mode with the `und process` command. The text file should contain one command per line. Omit the "und" from each command within a batch file. You can use # to begin comments. For example use either of the following commands to run the sequence in the `This.txt` file:

```
und process This.txt
und process This.txt MyDatabase.udb
```

The `This.txt` file might contain commands similar to these:

```
# My command file
c:\projects\MyDatabase.udb
settings -C++MacrosAdd VERSION="Option_2"
analyze      # update database
report       # generate reports
metrics      # generate metrics
```

- **Line mode:** You can specify a single command or set of commands on a single command line. You must specify the database to be used on each command line, because it is not remembered from line to line. Commands are run in the order they appear on the command line. The help and list commands cannot be combined with other commands. For example, you could run either of the following commands to create a database, add files, analyze all, and then exit:

```
und create -db c:\myDb.udb -languages c++ add @myFiles.txt analyze -all
und create -languages c++ add @myFiles.txt analyze -all c:\myDb.udb
```

is the equivalent of running the following set of commands in interactive mode:

```
create -languages c++ c:\myDb.udb
add @myFiles.txt
analyze -all
```

Alternately, you could run a sequence of line mode commands like the following:

```
und create -languages c++ c:\myDb.udb
und add @myFiles.txt c:\myDb.udb
und analyze -all c:\myDb.udb
```

In general, und commands are case-insensitive.

Und returns a value of 1 if an error occurred.

Und supports the following options that can be added to any command:

Option	Discussion
-db	Specify the database to use
-quiet	Print only errors. Do not print warnings or informational messages
-verbose	Print extra informational details.

Und accepts a number of separate commands. A different set of options is supported for each of these commands, and separate help is available for each. For example, for help on the add command, type:

```
und help add
```

The commands supported by und are as follows:

Option	Discussion	See
add	Adds files, directories, and roots	page 300
analyze	Analyzes the project files	page 303
codecheck	Runs CodeCheck	page 304
create	Creates an empty database	page 300
export	Exports settings, dependencies, or architectures	page 302
help	Gives help information for a command	page 300
import	Imports project settings and architectures	page 302
list	Lists information about the project	page 301
metrics	Generates project metrics	page 303
process	Runs all the commands in a text file in batch mode	page 298
purge	Purges the database	page 303
remove	Removes files, directories, roots, and architectures	page 301
report	Generates project reports	page 303
settings	Sets project settings and overrides	page 301
upperl	Runs Perl scripts	page 304
version	Shows the current software version	page 300

Refer to the sections that follow for details on the commands supported by und.

Getting Help on Und

Since we do frequent builds of *Understand*, it is likely that this manual may not describe all the options of the “und” command line. We recommend that you check the command-line help. For example, to get details on the report command, type:

```
und help report
```

You can see the version of *Understand* for the und command tool by using the following command:

```
und version
```

Creating a New Project

Use the und `create` command to create a new database (project). Specify the name of the database either with the `-db` option or as the last parameter. Any settings allowed with the `settings` command (see page 301) can also be used with `create`. For example:

```
und create -db newDB.udb -languages c++ c#
und create -open_files_as_read_only on newDB.udb
```

For more information, use the following command:

```
und help create
```

Adding Files to a Project

If you have a small number of source files then it may be easiest to just supply their names to the analyzer using the wildcarding abilities of your operating system shell. For example:

```
und -db myproject.udb add \usr\myproject
und -db myproject.udb add file1.cpp file2.cpp
und -db myproject.udb add *.cpp
```

In some cases, there may be too many file locations to use the `-add` technique. A common command line limitation is 255 characters. A directory with hundreds or thousands of files may easily exceed this limit. In this case, or when you want more fine-grained/repeatable control over what files are processed, you should create a “listfile”. This file must have a format of one filename per line:

```
c:\myfiles\myproject\myproject.c
c:\myfiles\myproject\myproject.h
c:\myfiles\myproject\support.c
c:\myfiles\myproject\io.c
c:\myfiles\myproject\io.h
h:\shared\allprojects\file2.c
h:\options\file3.c
h:\options\file4.c
h:\options\file5.c
. . .
```

You can then add all of these files as follows:

```
und -db myproject.udb add @myfiles.lis
```

Note that there is no limit on the number of files listed in the list file.

You can also use the `add` command to add named roots and Visual Studio projects. Options are available to set the watch behavior, subdirectory adding, the exclude list, file filtering, and languages.

For more information, use the following command:

```
und help add
```

Removing Items from a Project

Use the `und remove` command to remove files, directories, Visual Studio files, named roots, and architectures from a project.

Unless there is a name conflict, the type of item to be removed is automatically detected by `und`. If there is a conflict, the command defaults to deleting the directory with the specified name. You can use the `-file`, `-vs`, `-root`, and `-arch` options to override this default.

For example:

```
und remove someFile.cpp myProject.udb
und remove C:\SomeDirectory myProject.udb
und -db myProject.udb remove vs1.vcproj vs2.vcproj
und remove -file main.c myProject.udb
```

For more information, use the following command:

```
und help remove
```

Getting Information about a Project

Use the `und list` command to list file, setting, architecture, or named root settings in a project. For example:

```
und list -tree files myProject.udb
und list settings myProject.udb
und list arches myProject.udb
und list roots myProject.udb
```

There are a number of options for listing settings for the project. You can list all settings, language-specific settings, report settings, metric settings, include directories, macro definitions, and more. For example:

```
und list -override f1.cpp f2.java settings myDB.udb
und list -override @listfile.txt myDB.udb
und list -metrics -reports settings myDB.udb
und list -all settings myDB.udb
und list -lang C++ -macros -includes settings myDB.udb
und list -lang fortran settings myDB.udb
```

For more information, use the following command:

```
und help list
```

Modifying Project Settings

Use the `und settings` command to modify the settings in a project. You can find the names for each setting by using the following command:

```
und list -all settings myProject.udb
```

In general, setting names are the same as the field name in *Understand*, but with spaces omitted.

For example:

```
und settings -ReportDisplayCreationDate on myProject.udb
und settings -ReportFileNameDisplayMode full myProject.udb
und settings -ReportReports "Data Dictionary" "File Contents" myProject.udb
und settings -C++MacrosAdd MYLONG="Long Text" myProject.udb
und settings -ReportNumberOfPages 250 myProject.udb
```

For more information, use the following command:

```
und help settings
```

Importing into a Project

Use the `und import` command to import project settings or architectures from an XML file. In general, you might use this command when creating a new database to import setting that you have exported from another database.

For example:

```
und import settings.xml myNewProject.udb
und import -arch myArch.xml myProject.udb
```

For more information, use the following command:

```
und help import
```

Exporting from a Project

Use the `und export` command to export project settings, architectures, or a list of dependencies to an XML file.

For example, this command exports project settings to an XML file that you can use with the `und import` command:

```
und export toHere.xml myProject.udb
```

This command exports architectures to an XML file that you can use with the `und import` command:

```
und export -arch "Calendar" toHere.xml myProject.udb
```

These commands export file, architecture, and class dependencies to a CSV, matrix, or Cytoscape file. Several options are available to control the output of dependencies.

```
und export -dependencies file csv output.csv myProject.udb
und export -dependencies class matrix output.csv myProject.udb
und export -dependencies arch myArch csv output.csv myProject.udb
und export -dependencies -col refs -format short file csv out.csv myDB.udb
```

For more information, use the following command:

```
und help export
```

Analyzing a Project

Use the `und analyze` command to run (or rerun) the project analysis.

When you analyze a project, you have several options. You may re-analyze all files with the `-all` option (the default), only files that have changed with the `-changed` option, or a list of files with the `-files` option. For example:

```
und analyze myProject.udb
und analyze -files @someFile.txt
und -db myProject.udb analyze -rescan -changed
und analyze -files file1.cpp file2.cpp myProject.udb
```

You can scan project directories for new files with the `-rescan` option. (This is done automatically when you analyze all.)

If you are doing your first analysis after creating a new project, it doesn't matter which option you choose as it will parse all files regardless. However, if you are performing this function on a regular basis, you may prefer to do an incremental analysis where only the modified files and any other files dependent on those files are re-analyzed.

Use the `und purge` command to remove all parsed data from the Understand database, leaving only the project definition. This significantly shrinks the `udb` file size, which you may want to do before sharing the file or backing it up. Running the `analyze` command will repopulate the project. For example:

```
und purge myProject.udb
```

For more information, use the following command:

```
und help analyze
```

Generating Reports

Use the `und report` command to generate reports for the project. This command uses the current report settings, which can be viewed by using the `und list` command (see page 301), and changed using the `settings` command (see page 301). For example:

```
und list -reports settings myProject.udb
und report myProject.udb
```

Generating Metrics

Use the `und metrics` command to generate metrics reports for the project. You can generate project metrics (the default), architecture metrics, and the HTML metrics report. For example:

```
und metrics myProject.udb
und metrics -arch myArch myProject.udb
und metrics -html arch1 arch2 c:\temp myProject.udb
```

This command uses the current metrics settings, which can be viewed by using the `und list` command (see page 301), and changed using the `settings` command (see page 301). For example:

```
und list -metrics settings myProject.udb
```

For more information, use the following command:

```
und help metrics
```

Using CodeCheck

Use the `und codecheck` command to run the CodeCheck tool on the project and print the log to the screen. You need to provide the name of a CodeCheck configuration file and an output directory for the reports. For example:

```
und codecheck config.ini C:\temp myProject.udb
```

You can create a CodeCheck configuration file as described in *Using CodeCheck Configurations* on page 271.

Options are provided to specify which files to run the CodeCheck configuration on, whether to show ignored violations, whether to flatten the directory tree, and whether to generate HTML output in addition to the default CSV output. For example:

```
und codecheck -html -files filelist.txt config.ini C:\temp myProject.udb
```

For more information, use the following command:

```
und help codecheck
```

Running Perl Scripts

Use the `und uperl` command to run Perl scripts from the command line. For example, the following command would run the `myScript.pl` file with the `arg1` space and `arg2` arguments passed to Perl:

```
und uperl myScript.pl -quiet "arg1 space" arg2 myProject.udb
```

For more information, use the following command:

```
und help uperl
```

Creating a List of Files

Where a command accepts a `@lisfile.txt` for an option, the file must contain one item per line. Full or relative paths may be used. Relative paths are relative to the current directory. A `#` sign in the first column of a line in the file indicates a comment. If an item has a definition, for example a macro definition, the macro name and its value must be separated by an `=` sign. For example, `DEBUG=true`.

On UNIX here are a couple ways to create such a file:

- Use the `'ls'` command, as in:

```
ls *.c *.h > my_project.txt
```

- Use the `'find'` command to recurse subdirectories, as in:

```
find . -name "*.c *.h" -print > my_project.txt
```

In a Windows command shell:

- Use the `dir` command with the `/b` option:

```
dir /b *.c *.h > my_project.txt
```

- Use the `/s` option to recurse subdirectories, as in:

```
dir /b /s *.c *.h > my_project.txt
```

Using the understand Command Line

The *Understand* GUI is launched by the “understand” executable. Normally, you launch this using the shortcuts provided by the installation. If you like, you can modify this using the following command-line syntax.

```
understand [file_1 ... file_n] [-options]
```

Any filenames listed on the command line are opened along with The *Understand* GUI. For example:

```
understand source.c source.h -db myproject.ldb
```

The available command-line options (also called command-line switches) are as follows:

Option	Discussion
-contextmenu <i>filename</i> [-line # -col # -text #]	Shows the context (right-click) menu for the specified filename at the mouse location. Optionally shows the context menu for the entity located at -line -col (The -text option provides a name hint for the entity).
-cwd <i>path</i>	Set the current working directory to "path". This takes precedence over the last working directory for a project loaded with -db or -lastproject.
-db <i>filename</i>	Open the project specified by the filename.
-diff <i>left_path right_path</i>	Compare the two specified files or folders as with the Tools > Compare command within <i>Understand</i> .
-existing	Detects any running instance of <i>Understand</i> and sends the command line to that instance.
-importusertools <i>importfile.ini</i>	Import user tool definitions from an initialization file.
-lastproject	Open the last project opened by the application.
-lastproject_cwd	Use the directory of the last opened project as the current working directory.
-new	Force the creation of a new instance of <i>Understand</i> . If you use the operating system to open a file with an extension that opens <i>Understand</i> , by default that file opens in any existing instance. You can use this command-line option to force a new instance to open.
-noproject	Ignore all project load requests on startup. (This also clears the "Open Last Project" application setting.)
-no_splashscreen	Use this option to skip the splash screen when <i>Understand</i> starts up. This setting is stored until you change it in the Tools > Options dialog.
-quiet_startup	Use this option to disable all dialogs and splash screens shown during startup.
-SlowConnect	Allow for a longer timeout period when communicating with the license server.

Option	Discussion
-visit <i>filename</i> [<i>line# column#</i>]	Open the file "filename" in an editor window. Optionally position the cursor at the specified line number and column number in the specified file.
-wait	When used with the -existing option, causes this instance of <i>Understand</i> to block while waiting for the other instance to finish the given command.

Using Buildspy to Build Understand Projects

Buildspy is a tool that allows gcc/g++ users to create an *Understand* project during a build. Buildspy gets lists of files, includes, and macros from the compiler. This can save time and improve project accuracy.

To use Buildspy, follow these steps:

- 1 Change the compiler command from `gcc/g++` to `gccwrapper/g++wrapper` in your makefile or build system.
- 2 Either add the `<SciTools>/bin/<platform>/buildspy` directory to your PATH definition or use the full path to the `gccwrapper/g++wrapper` executables in your makefile or build system. On Linux, this might be the `/SciTools/bin/linux32/buildspy` directory. On Windows, this might be the `C:\Program Files\SciTools\bin\pc-win64\buildspy` directory.
- 3 Perform a `make clean` or equivalent command.
- 4 Run a command similar to the following:

```
buildspy -db path/name.udb -cmd <compile_command>
```

For example:

```
buildspy -db -/Documents/MyProject.udb -cmd make
```

- 5 When the build has finished running, open the *Understand* project that was created and choose **Project > Analyze All Files**.

The `buildspy` command sends information from `gccwrapper/g++wrapper` to Buildspy, which allows it to build a complete *Understand* project. The wrappers then call the corresponding compiler.

To change the compiler run by `gccwrapper` or `g++wrapper`, edit the configuration file located at `$HOME/.config/SciTools` on Linux systems and `$HOME/Library/Preferences` on Mac. The wrappers will work with any compiler that has gcc-like syntax.

This chapter lists of commands provided by *Understand*. These lists provide cross references to information about these commands in this manual.

Since new versions of *Understand* are provided frequently, these lists are subject to change.

This chapter contains the following sections:.

Section	Page
File Menu	308
Edit Menu	309
Search Menu	309
View Menu	310
Project Menu	310
Reports Menu	311
Metrics Menu	311
Graphs Menu	312
CodeCheck Menu	312
Annotations Menu	312
Tools Menu	313
Window Menu	314
Help Menu	314

File Menu

The **File** menu in *Understand* contains the following commands:

Command	See
New > Project	page 35
New > File	page 165
Open > Project	page 21
Open > File	page 165
Open > Understand 1.4 Project	page 119
Close <project_name>	page 21
Export to Image File	page 255
Save Configuration	page 271
Save <filename>	page 162
Save <filename> As	page 162
Save All	page 162
Page Setup	page 174
Print <filename>	page 174
Print Entity Graph	page 257
Recent Files	page 97
Recent Projects	page 97
Exit	page 51

Edit Menu

The **Edit** menu in *Understand* contains the following commands:

Command	See
Undo	page 161
Redo	page 161
Cut	page 160
Copy	page 160
Copy Image to Clipboard	page 255
Paste	page 160
Select All	page 160
Comment Selection	page 164
Uncomment Selection	page 164
Change Case	page 164
Toggle Overtyping	page 165
Bookmarks	page 166

Search Menu

The **Search** menu in *Understand* contains the following commands:

Command	See
Find	page 155
Find Previous	page 155
Find & Replace	page 155
Go to Line	page 159
Go to Matching Brace	page 163
Instant Search	page 141
Find in Files	page 143
Replace in Files	page 146
Show Find Results	page 145
Find Entity	page 148

View Menu

The **View** menu in *Understand* contains the following commands:

Command	See
Toolbars	page 154
Browse Mode	page 160
Zoom	page 158
Fold All	page 163
Soft Wrap	page 165
Hide Inactive Lines	page 163
Bookmarks	page 166
Contextual Information	page 156
Entity Filter	page 123
Entity Locator	page 148
Information Browser	page 125
Favorites	page 136
Last Parse Log	page 114
Project Browser	page 130
Scope List	page 159
Window Selector	page 152

Project Menu

The **Project** menu in *Understand* contains the following commands:

Command	See
Configure Project	page 39
Rescan Project Directories	page 46
Analyze Changed Files	page 114
Analyze All Files	page 114
Project Overview Charts	page 229
Architectures > New Architecture	page 185
Architectures > Browse Architectures	page 177
Architectures > Manage Architectures	page 184

Reports Menu

The **Reports** menu in *Understand* contains the following commands:

Command	See
Configure Reports	page 191
Generate Reports	page 193
View Reports > HTML	page 194
View Reports > Text	page 194
Dependency > Architecture Dependencies	page 225
Dependency > File Dependencies	page 225
Dependency > Class Dependencies	page 225
Project Interactive Reports	page 32

Metrics Menu

The **Metrics** menu in *Understand* contains the following commands:

Command	See
Metrics Summary	page 215
Browse Metrics	page 216
Export Metrics	page 218
Project Metric Charts > Code Volume	page 220
Project Metric Charts > File Volume	page 220
Project Metric Charts > Average Complexity	page 220
Project Metric Charts > Sum Complexity	page 220
Configure Metric Charts	page 220
Metrics Treemap	page 223

Graphs Menu

The **Graphs** menu in *Understand* contains the following commands:

Command	See
Dependency Graphs > By <architecture>	page 179
Dependency Graphs > Load Saved Dependency Graph	page 180
Project Graphs	page 32
Graphs for <selected entity>	page 231

CodeCheck Menu

The **CodeCheck** menu in *Understand* contains the following commands:

Command	See
Open CodeCheck	page 261
Re-Run Previous Checks	page 271
Standards	page 271
Saved Configurations	page 271
Implement Your Corporate Standard	page 272

Annotations Menu

The **Annotations** menu in *Understand* contains the following commands:

Command	See
Annotate	page 168
Filter Annotations	page 172
Manage Annotations	page 173
Search Annotations	page 171
Annotation Options	page 59
Refresh Annotations	page 59
Display Inline	page 59
Display Hover	page 59
Display Indicator	page 59

Tools Menu

The **Tools** menu in *Understand* contains the following commands:

Command	See
Run Command	page 295
User Tools > <tool_name>	page 292
User Tools > Configure	page 284
Editor Macros > Record Macro	page 165
Editor Macros > Replay Macro	page 165
Scheduler > Scheduled Activities	page 50
Compare > Compare Files/Folders	page 275
Compare > Compare Entities	page 277
Compare > Compare Arbitrary Text	page 278
Options	page 93

Window Menu

The **Window** menu in *Understand* contains the following commands:

Command	See
Close <current_file>	page 162
Close All Document Windows	page 152
Release Window	page 152
Split Vertically	page 152
Split Horizontally	page 152
Unsplit	page 152
Tile	page 152
Cascade	page 152
Predefined Windows Layouts	page 152
<open source file list>	page 152
Windows	page 152

Help Menu

The **Help** menu in *Understand* contains the following commands:

Command	See
Help Content	page 16
Key Bindings	page 100
Example Projects	page 16
PERL API Documentation	page 32
Python API Documentation	page 32
Frequently Asked Questions	page 16
Getting Started	page 21
Licensing	page 14
View SciTools Blog	page 16
Check for Updates	page 21
About Understand	page 16

Index

Symbols

^ in regular expressions 151
: in F77 identifiers, FORTRAN 81
? in regular expressions 151
? wild card 150
. in regular expressions 151
" prefixing octal constants or string literals, FORTRAN 81
"" surrounding normal includes 72
[-] in regular expressions 151
[] in regular expressions 151
[^] in regular expressions 151
* in regular expressions 151
* wild card 150
/* ... */ C-style comments 69, 81
\ in regular expressions 151
\< in regular expressions 151
\> in regular expressions 151
comment in command line file 304
+ expanding tree in Information Browser 126
+ in regular expressions 151
<> surrounding system includes 72
= macro definition in command line 304
| in regular expressions 151
\$ in regular expressions 151
\$ prefixing external tool parameters 285, 286

A

About Understand option, Help menu 16
absolute portability mode 47
Activate when Control key is pressed field, Browse Mode Editor options 111
actual parameters, relationship to formal parameters, Ada 62
Ada
 configuration for 61
 macro definitions 63
 versions supported 15
Ada category, Project Configuration dialog 61
Add found include files to source list field, C++ Includes 71, 78
Add found system include files to source list field, C++ Includes 71, 78

-addDir option, und command 300
-addFiles option, und command 300
Adobe Acrobat, saving graphical views to 258
Advanced category, Editor options 107
Aggregate Nodes by options 255
Alerts category, User Interface options 98
Allow Colons in Names field, FORTRAN 81
Allow C-style comments field, FORTRAN 81
Allow embedded SQL field, Pascal 87
Allow function declaration without parentheses field, FORTRAN 81
Allow interactivity during intensive processing field, General options 95
Allow Nested Comments field, C++ 69
Allow quote in octal constants field, FORTRAN 81
Analysis Options area 285
Analyze All Files option, Project menu 51, 114
Analyze category, Understand Options dialog 101
Analyze Changed Files option, Project menu 114
-analyze option, und command 303
-analyzeFiles option, und command 303
analyzing projects 114
 after changing configuration of 40
 beep on completion of 101
 on command line 303
 scheduling 51
and operators, including in strict complexity, Ada 62
angle brackets (<>) surrounding system includes 72
Animate windows/drawers field, User Interface options 96
Annotations category, Project Configuration dialog 59
Annotations menu 312
anti-virus software, turning off while generating reports 193
Apache Lucene syntax 141
APIs 32
Append the names of externally linkable entities with field
 C++ 70
 FORTRAN 81
Application font field, General options 94
Architect Manager 178, 184
Architecture Browser 30, 177
 duplicating architectures 178, 185
 graphical views in 178, 179
 metrics in 178, 183
 opening Architect Manager from 178, 184

- opening Architecture Builder from 178
 - renaming architectures 178
 - right-clicking in 178
 - XML listing in 178
 - Architecture Builder 178, 187
 - Architecture Graph View 240
 - Architecture Wizard 186
 - architectures 19, 176
 - auto-architectures 177
 - creating 185
 - duplicating 178, 185
 - editing. *See* Architecture Builder
 - exporting dependency list 225
 - graphical views of 178, 179
 - hierarchies of, exploring 177
 - hierarchies of, graphical view of 240
 - list of 177
 - listing 30
 - managing. *See* Architect Manager
 - metrics about 178, 183
 - metrics graphs about 220
 - navigating 30
 - renaming 178, 185
 - XML listing of 178, 189
 - arrows, for Information Browser history 130
 - ASCII text. *See* text
 - ASP style tags 92
 - assembly
 - configuration for 65
 - embedded in C code 70, 74
 - include files, directories for 65
 - Assembly category, Project Configuration dialog 65
 - asterisk (*)
 - in regular expressions 151
 - wild card 150
 - Auto Category, C++ Includes 72
 - auto-architectures 30, 177
 - Auto-complete fields, Advanced Editor options 108
 - Auto-indent fields, Advanced Editor options 108, 109
 - Automatic compool file field, JOVIAL 85
 - Average Complexity metrics graph 220
- B**
- background processing, interactivity during 95
 - backslash (\) in regular expressions 151
 - base classes
 - count of 210
 - displaying 237, 239, 244
 - Base Classes View 237, 239
 - beep on parse completion 101
 - bitmaps, saving graphical views as 255
 - black text 110
 - blank lines
 - in Entity Comparison area 281
 - Blank metric 212
 - blocks
 - expanding and collapsing 158, 163
 - in Filter Area 25
 - blue italic text 110
 - blue text 110
 - bookmarks
 - creating 166
 - list of 154, 166
 - navigating 166
 - Bookmarks option, Edit menu 166
 - Bookmarks option, View menu 154, 166
 - Boolean searches 142
 - braces, matching. *See* brackets
 - brackets
 - in regular expressions 151
 - matching 163
 - Browse Architectures option, Project menu 177
 - Browse Metric Charts option, Metrics menu 224
 - Browse Metrics option, Metrics menu 214, 216
 - Browse Mode option, View menu 160
 - Browse Mode, Source Editor 160
 - Browser Area 20
 - Build Log 34
 - Buildspy 306
 - Butterfly View 237, 238
 - Butterfly-Dependency graph 179
- C**
- C/C++
 - API for custom reports 32, 196, 214
 - configuration for 69, 76
 - include files, auto including 72
 - include files, directories for 71, 77
 - include files, ignoring 72
 - macro definitions 73, 79
 - preprocessor directives 73
 - strict analysis 76
 - versions supported 15

C#
 configuration for 79
 reference files, importing 79
 versions supported 15
C# category, Project Configuration dialog 79
C++ (Strict) category, Project Configuration dialog 76
C++ category, Project Configuration dialog 69
 caching include files, C++ 70
 Calendar auto-architecture 30, 177
 Called by menu, graphical views 246
 Called By View 237, 238
 Calls View 237, 238
 capitalization, of selected text 164
 Captured output font field, Command Window options 102
 caret (^) in regular expressions 151
 Caret Line field, Editor options 106
 Cascading Style Sheet 193
case
 changing for selected text 164
 of displayed entity names, Ada 62
 of displayed entity names, FORTRAN 81
 of displayed entity names, JOVIAL 85
 of displayed entity names, Pascal 88
 of displayed entity names, PL/M 89
 of externally linkable entities, Ada 62
 of externally linkable entities, FORTRAN 81
 of externally linkable entities, Pascal 88
 of identifier names, FORTRAN 81
Case of externally linkable entities field
 Ada 62
 FORTRAN 81
 Pascal 88
Case sensitive identifiers field, FORTRAN 81
case sensitivity
 of Entity Comparison area 281
 of FORTRAN identifiers 81
CBO (Count of Coupled Classes) metric 210
Change Case menu option 164
Change Case option, Edit menu 164
Change Log 34
Changed Entities area, Change Results window 279
 character strings, red text for 110
 Check for Updates option, Help menu 22
 Child Lib Units View 237
 children of entity 237
CIS. See Contextual Information Sidebar
Class and Interface Cross-Reference report 199
Class Declaration View 240, 244
class diagram 242
Class Extend Tree report 203
.class files 83
Class Inheritance View 237
Class Metrics report 210
Class OO Metrics report 210
Class Paths category, Java 83
classes
 base classes for 210, 237, 239, 244
 cohesion of 210
 coupled 210
 declaration for 244
 derived 210, 237, 239
 exporting dependency list 226
 extended by other classes 237, 240, 247
 extending other classes 237, 239, 247
 as hexagons, in graphical views 245
 implemented 247
 implemented by 248
 information about 240
 inherited from other classes 237
 listing in Filter Area 25
 metrics about 210
 providing without source code, Java 83
 reports about 199, 203, 210
clipboard 280
 copying graphical view to 256
 copying text from Source Editor to 160
Close All Document Windows option, Window menu 152, 162
Close option
 File menu 22
 Window menu 152, 162
Close Selected Window(s) command 154
cluster graphs 254
 customizer 180
COBOL
 configuration for 67
 copybook files, directories for 67
COBOL category, Project Configuration dialog 67
Code metric 212
Code Volume metrics graph 220
CodeCheck 259
 checks 262
 configuration 263, 271

- ignoring violations 270
- result log 264
- running 261
- scripts 272
- CodeCheck menu 312
- cohesion for class data members 210
- colon (:) in F77 identifiers, FORTRAN 81
- Color Mode field, Advanced Editor options 107
- colors
 - in Entity Comparison area 281
 - in file/folder comparisons 276
 - of highlighted differences in Entity Comparison area 280
 - in HTML reports 193
 - of merged differences in Difference List area 282
 - of merged differences in Entity Comparison area 282
 - for printing source code 107
 - of rows in User Interface, alternating 96
 - in Source Editor, customizing 158, 110
 - in Source Editor, default 110
- column truncation
 - FORTRAN 81
 - JOVIAL 85
- command line. *See* und command; understand command
- command renames, reports about 204
- Command window 295
- Command Window category, Understand Options dialog 102
- commands, external. *See* external tools
- Comment metric 212
- Comment Selection menu option 164
- comments
 - adding or removing 164
 - associating with entities, Ada 62
 - associating with entities, C 77
 - associating with entities, C++ 70
 - associating with entities, Java 83
 - blue italic text for 110
 - C-style 69
 - nested, C++ 69
- companion file 165
- Compaq Pascal 87
- Compare Arbitrary Text option, Tools menu 278
- Compare Entities option, Tools menu 277
- Compare Files/Folders option, Tools menu 275
- compilation environment, Ada 62
- compilation unit
 - Child Library Units of 237
 - declaration nesting of program units in 237
 - entities called by 237
 - instantiation tree for 237
 - With By relationships of 237, 253
 - With relationships of 237
- compiler
 - COBOL 67
 - compared to Understand 34
- Compiler field
 - C++ 69
 - PL/M 89
- Compiler Include Paths field, C++ 69
- complexity
 - exception handlers included in, Ada 62
 - FOR-loops included in, Ada 62
 - metrics graphs about 220
 - strict, and/or operators in, Ada 62
- Component field, Key Bindings options 100
- compool file, JOVIAL 85
- Configure category, Understand Options dialog 102
- Configure Metric Charts option, Metrics menu 214, 220
- Configure option, Project menu 39
- Configure Reports option, Reports menu 191
- constants
 - displayed in graphical views 246
 - reports about 207
- Constants menu, graphical views 246
- contact information 15
- Context Browser, Contextual Information Sidebar 156
- contextmenu option, understand command 305
- Contextual Information option, View menu 156
- Contextual Information Sidebar (CIS) 156
- Control Flow View 240, 244
- !COPY directives, directories to search, JOVIAL 86
- Copy category, JOVIAL 86
- Copy Hover Text menu option 162
- Copy option, Edit menu 160
- Copybook category 67
- copybook files
 - adding to project, COBOL 67
- copying text 280
 - rectangular area 160
- Count and/or operators in strict complexity field, Ada 62
- Count exception handlers in complexity field, Ada 62
- Count for-loops in complexity field, Ada 62

Count of All Methods metric 210
Count of Base Classes metric 210
Count of Coupled Classes metric 210
Count of Derived Classes metric 210
Count of Instance Methods metric 210
Count of Instance Variables metric 210
Count of Methods metric 210
coupled classes 210
Create and cross-reference record object components field, Ada 62
Create implicit special member functions field 70
Create references in inactive code field, C++ 70
Create references in inline assembly, C++ 70
Create references to local objects field, C++ 70
Create references to macros during macro expansion field, C++ 70
Create references to parameters field, C++ 70
Create relations between formal and actual parameters field, Ada 62
Crossing layout option 249
cross-reference reports 197
cross-referencing record object components, Ada 62
CSS files 15
C-style comments
 in FORTRAN 81
 nesting of, allowing 69
CSV file
 exporting dependencies to 225
 exporting metrics to 51, 214, 218
 exporting project overview charts to 230
Ctrl+Alt+F keystroke, find and replace text 155
Ctrl+F keystroke, find text 155
Ctrl+j keystroke, jump to matching bracket 163
Ctrl+right-click keystroke
 creating new windows 23, 121
Ctrl+Shift+j keystroke, select text in brackets 163
\$Cur variables 285, 286
Cut option, Edit menu 160
-cwd option
 understand command 305
Cyclomatic complexity 205, 212
Cytoscape
 installation location 104
 XML output 225
Cytoscape XML 225, 226

D

dashed outline shapes, in graphical views 245
Data Dictionary report 197
Data Members View 240
database 19, 34
 changes to format of 34
 creating on command line 298
 file extension for 19, 34
 multi-user read/write access for 34
 See *also* project
-db option
 understand command 305
DEC Pascal 87
\$Decl variables 285, 286
Declaration File View 240, 243
Declaration Tree report 202
Declaration Tree View 237
Declaration Type View 240, 243
Declaration View 240, 241
declaration views
 calling methods displayed in 246
 constants displayed in 246
 default members displayed in 246
 extended by classes displayed in 247
 extended classes displayed in 247
 external functions displayed in 247
 file dependencies displayed in 246
 globals displayed in 247
 header files include by's displayed 248
 implemented by classes displayed in 248
 implemented classes displayed in 247
 imported entities displayed in 248
 include files displayed in 248
 inherited entities displayed in 248
 invocations displayed in 249
 local items displayed in 250
 members displayed in 250
 objects displayed in 251
 operators displayed in 250
 private members displayed in 251
 protected members displayed in 251
 public members displayed in 251
 rename declarations in 251
 static functions displayed in 252
 types displayed in 253
 used-by items shown in 253

- variables displayed in 253
 - With By relationships displayed in 253
 - With relationships displayed in 253
 - See also graphical views
 - declarations
 - implicit, report about 207
 - local, including in database, C++ 70
 - Declared In View 237
 - Default Members menu, graphical views 246
 - Default style field, Editor options 105
 - default working directory 95
 - defines. See macros
 - Delphi Pascal 87
 - Depended On By graph 179, 237
 - dependencies
 - browsing 133
 - exporting 134, 225
 - file 246
 - Dependency Browser 133
 - Dependency category, Understand Options dialog 104
 - Dependency Graph 178
 - Dependency Graphs option, Graphs menu 179
 - Dependency menu 225
 - Dependent menu
 - graphical views 246
 - Dependent Of menu, graphical views 246
 - Depends On graph 179, 237
 - Depth menu, graphical views 246
 - derived classes 210, 237, 239
 - Derived Classes View 237, 239
 - DFM converter exe field 87
 - diff option, understand command 305
 - Difference List area, Change Results window 279, 282
 - Dim highlight color field 96
 - directives, C++ 73
 - directories
 - adding to project 43
 - comparing 275
 - copying 276
 - deleting from project 45
 - overriding settings for 45
 - watched, scanning 46, 50
 - watched, setting 44, 45
 - Directory Structure auto-architecture 30, 177
 - Display entity names as field
 - Ada 62
 - FORTTRAN 81
 - JOVIAL 85
 - Pascal 88
 - PL/M 89
 - DIT (Max Inheritance Tree) metric 210
 - Dock Window Layouts field, User Interface options 97
 - docking windows 18
 - Document Area 20
 - documentation 16, 21
 - dollar sign (\$)
 - in regular expressions 151
 - prefixing external tool parameters 285, 286
 - DOS line termination style
 - for reports 193
 - for saving source files 106
 - DOT files, saving graphical views as 256
 - double quotes. See quotes
 - double-click merging, in Entity Comparison area 281, 282
 - double-clicking
 - in Bookmarks area 166
 - column header dividers 148
 - entities in Entity Filter 128
 - entities in graphical view 233
 - entities in Information Browser 128
 - in Exploring View 132
 - in Find Results window 145
 - messages in Parse Log window 115
 - in Project Browser 216
 - in Source Editor 29
 - drawers 18, 96
 - Drill down command 225
 - drop-down menu See right-clicking
 - .dsp file extension 57
 - .dsw file extension 57
 - Duplicate Architecture menu option 178
 - duplicate references, C++ 70
 - Duplicate Subtrees menu, graphical views 246
- ## E
- Edges Shown options 254
 - edges, in dependency graphs 180
 - Edit Architecture menu option 178, 187
 - Edit Graphic Filters menu option 235
 - Edit menu 309
 - Bookmarks option 166
 - Change Case option 164
 - Copy option 160

-
- Cut option 160
 - Select All option 160
 - Toggle Overtyping option 165
 - Editor category, Understand Options dialog 105
 - Editor Macros option, Tools menu 165
 - editor windows, saving automatically 94
 - editor, external 112
 - editor, source. *See* Source Editor
 - Emacs editor 112
 - embedded SQL, in Pascal 87
 - Enable Editor Tooltips field 111
 - Enable permissions checking for NTFS filesystems field 95
 - encoding formats 49, 105
 - entities 19
 - comments associated with, Ada 62
 - comments associated with, C 77
 - comments associated with, Java 83
 - comparing two entities 277
 - comparisons between versions of 279
 - components of 240
 - current, information about 156
 - declaration structure for 240, 241
 - displaying source for. *See* Source Editor
 - favorites of, displaying 136
 - favorites of, marking 135
 - full name of, displaying 124
 - graphical views of. *See* graphical views
 - hierarchy of, exploring 132, 156
 - hierarchy of, graphical view 232, 237
 - information about. *See* Information Browser
 - list of, alphabetic 197
 - listed in Entity Filter 123
 - listed in Entity Locator 26, 148
 - listed in Filter Area 25
 - metrics for. *See* metrics
 - names of, formatting for reports 55
 - as rectangles, in graphical views 245
 - references for 128
 - relationships between 19
 - renamed 241
 - reports about. *See* reports
 - sorting of 123
 - structural information about, graphical view of 233, 240
 - unknown 253
 - unresolved 253
 - XML listing of 178
 - See also specific entities*
 - Entity Comparison area, Change Results window 279
 - Entity Filter 123
 - displaying information about entities in 125
 - displaying source of selected entity 128
 - entities not listed in 26, 148
 - jumping to entities in 25
 - location of 20
 - right-clicking in 24
 - Entity Graph option, Window menu 233
 - entity index, for HTML reports 194
 - Entity Locator 26, 148
 - column headers in, customizing 149
 - columns in, hiding and reordering 149
 - columns in, resizing 148
 - filtering by selection 149
 - filtering manually 150
 - filtering with regular expressions 150
 - filtering with wildcards 150
 - length of names in 148
 - opening 148
 - right-clicking in 26, 148
 - right-clicking on column header 149, 150
 - right-clicking on entities 149
 - sorting entities in 149
 - Entity Locator option, View menu 26, 148
 - entity_index.html file 194
 - environment variables
 - using in include paths, assembly 66, 68
 - using in include paths, C++ 71, 78
 - equal sign (=) in macro definition in command line 304
 - errors
 - displaying from Parse Log window 115
 - parse errors, prompting for, Ada 63
 - parsing, FORTRAN 81
 - Essential metric 212
 - Example Projects option, Help menu 16
 - Exception Cross-Reference report 201
 - exceptions
 - handlers for, including in Ada complexity 62
 - reports about 201
 - existing option, understand command 305
 - Expand Recursive Nodes menu, graphical views 247
 - Expand Repeated Nodes menu, graphical views 247
 - Exploring view 132
 - Export Dependency CSV 225
-

- Export Metrics option, Metrics menu 51, 214, 218
 - exporting tool commands 294
 - Extended By menu, graphical views 247
 - Extended By View 237, 240
 - Extends menu, graphical views 247
 - Extends View 237, 239
 - extensions. *See* file extensions
 - external editor 112
 - External Editor category, Editor options 112
 - External Functions menu, graphical views 247
 - external tools
 - adding to Right-click Menu 285, 291
 - adding to toolbar 293
 - adding to User Tools menu 292
 - commands for, importing and exporting 294
 - commands for, running 295
 - configuring 284
 - editor 112
 - externally linkable entities
 - case of, Ada 62
 - case of, FORTRAN 81
 - case of, Pascal 88
 - prefix for, C++ 70
 - prefix for, FORTRAN 81
 - suffix for, C++ 70
 - suffix for, FORTRAN 81
 - Externally Modified Files field, Editor options 106
- F**
- F5 key, Find in Files option 143
 - FAQ option, Help menu 16
 - favorites 27
 - displaying 136
 - marking entities as 135
 - Favorites option, View menu 27, 136
 - File Average Metrics report 212
 - File Contents report 198
 - file extensions
 - configuring for project 48
 - for database 19, 34
 - for MSVC project files 57
 - File Information, Contextual Information Sidebar 156
 - File menu 308
 - Close option 22
 - New > File option 165
 - New > Project option 35
 - Open > File option 165
 - Open > Project option 21
 - Open > Understand 1.4 Project option 119
 - Page Setup option 174
 - Print Entity Graph option 257
 - Print option 174
 - Recent Projects option 21
 - Save All option 162
 - Save option 162
 - File Metrics report 211
 - File Mode field, Editor options 105
 - File Options category, Project Configuration dialog 49
 - file permission checking 95
 - File Sync box 127
 - File Types category, Project Configuration dialog 48
 - File Volume metrics graph 220
 - Filename menu, graphical views 247
 - filenames
 - in graphical views 247
 - for reports 194
 - in title areas 97
 - files
 - comparing 275
 - copying 276
 - exporting dependency list 225
 - Information Browser for 156
 - location included in listing of 124
 - searching 27, 143
 - searching and replacing in 146
 - toolbar for 154
 - See also* database; header files; include files; MSVC project; project; source files
 - Files category, Project Configuration dialog 42
 - Filter Area 25
 - See also* Entity Filter; Project Browser
 - Filter By Selection menu option 149
 - Filter field, Entity Filter 124
 - filters
 - for graphical views 235
 - See also* Entity Filter; Entity Locator
 - Find dialog 155
 - Find Entity option, Search menu 148
 - Find in Files dialog 20, 27
 - Find in Files menu option 27, 143
 - Find option, Search menu 155
 - Find Results window 27, 145
 - fixed file format, FORTRAN 80

flow chart symbols 245
 flow charts 240, 244
 Fold All option, View menu 163
 fold icon 279
 folders. *See* directories
 folding code 163
 font
 in Command window, changing 102
 for printing source code 107
 for Source Editor windows 105
 in Source Editor windows 167
 in Understand, changing 94
 Font Size field, Advanced Editor options 107
 FOR-loops, including in complexity metrics, Ada 62
 formal parameters, relationship to actual parameters,
 Ada 62
 Format field, FORTRAN 80
 FORTRAN
 configuration for 80
 extensions supported 15
 include files 82
 macro definitions 82
 reports showing non-standard extension usage 206
 versions supported 15
 FORTRAN category, Project Configuration dialog 80
 FORTRAN Extension Usage report 206
 frames in windows, sliding 18
 Frameworks Category, C++ Includes 78
 free file format, FORTRAN 80
 function declarations without parentheses, FORTRAN
 81
 Function Pointer menu, graphical views 247
 functions
 external 247
 in graphical views 251
 listing in Filter Area 25
 metrics about 212
 reports about 198, 212
 static 252
 See also invocations; procedures; program units
 fuzzy search 142

G

gcc build 306
 General category, Understand Options dialog 94
 Generate Reports option, Reports menu 193
 Generic Instantiation report 204
 generic unit
 instantiation hierarchy for 237
 instantiation tree for 237
 Getting Started dialog
 displaying at startup 94
 opening 22
 Getting Started option, Help menu 16, 22
 Getting Started tab 21
 Globals menu, graphical views 247
 Go To Line dialog 160
 Go to Matching Brace option, Search menu 163
 Graph Architecture 178, 179
 Graph Architecture graph 179
 Graph Customizer pane 179
 Graph View of graphs 229
 Graph View, for metrics graphs 222
 Graphic Filter dialog 235
 graphical user interface (GUI), parts of 20
 graphical views 31, 229
 of architecture 178, 179
 browsing 233
 calling methods displayed in 246
 constants displayed in 246
 copying to clipboard 256
 default members displayed in 246
 diagram of 20
 displaying 231
 entity name truncation for 252
 expanding or contracting nodes in 234
 extended by classes displayed in 247
 extended classes displayed in 247
 external functions displayed in 247
 file dependencies displayed in 246
 filename display options 247
 filtering 235
 fullnames displayed in 250
 function pointers displayed in 247
 functions displayed in 251
 globals displayed in 247
 Graph View of 229
 header files include by's displayed 248
 hierarchy levels displayed in 250
 hierarchy views 232, 237
 implemented by classes displayed in 248
 implemented classes displayed in 247
 imported entities displayed in 248

- include files displayed in 248
- inherited entities displayed in 248
- intrinsic functions displayed in 248
- invocations displayed in 249
- layout configuration for 235, 245, 249
- local items displayed in 250
- members displayed in 250
- multiple subtrees displayed in 246
- objects displayed in 251
- open, list of 233
- operators displayed in 250, 251
- parameters, displaying in 251
- path highlighting in 234
- printing 234, 257
- private members displayed in 251
- procedures displayed in 251
- protected members displayed in 251
- public members displayed in 251
- rename declarations in 251
- reusing 99, 236
- saving 234, 255
- saving as PDF 258
- scrolling in 233
- sorting entities in 252
- spacing entities in 252
- SQL entities shown in 252
- static functions displayed in 252
- structure views 233, 240
- synchronizing with other windows 236
- Table View of 230
- text size of 252
- types displayed in 253
- unknown entities, displaying 253
- unresolved entities, displaying 253
- used-by items shown in 253
- uses items shown by 253
- variables displayed in 253
- With By relationships displayed in 253
- With relationships displayed in 253
- zooming 234
- See *also* structure views; hierarchy views
- Graphical Views menu option 178, 179, 231
- Graphs category, Understand Options dialog 113
- Graphs for option, Graphs menu 179, 231
- Graphs menu 312
 - Dependency Graphs option 179
 - Graphs for option 179, 231

- Project Graphs option 32, 196
- graphs, of metrics 220
- gray background for text 111
- gray shapes, in graphical views 245
- green text 110
- GUI (graphical user interface), parts of 20
- GVIM editor 112

H

- header files 248
- help 16, 21
- Help menu 314
 - About Understand 16
 - Check for Updates option 22
 - Example Projects option 16
 - FAQ option 16
 - Getting Started option 16, 22
 - Help Content option 16
 - Key Bindings option 16, 165
 - Licensing option 14
 - Perl API Documentation option 16, 32, 214
 - Python API Documentation option 16, 32
 - View SciTools Blog option 16
- help option, und command 300
- hexagons, in graphical views 245
- Hide Inactive Lines option, View menu 163
- hiding common lines, in Entity Comparison area 281
- hierarchy views 31, 232
 - layout of 249
 - list of 237
 - multiple subtrees displayed in 246
 - number of levels in 250
 - parameters, displaying in 251
 - See *also* graphical views
- highlighting full line at cursor 106
- history
 - Information Browser 126, 130
 - Source Editor 153
 - toolbar for 154
- Horizontal Non-Crossing layout option 249
- hover menu 162
- hover text, copying 162
- HTML
 - colors in reports 193
 - entity index in reports 194
 - exporting metrics to 51, 214, 217
 - files 15

generating reports as 32, 54, 191, 193
viewing reports as 194
Hyper Grep. See Find in Files dialog
hyphen (-) collapsing tree in Information Browser 126

I

IB. See Information Browser
IFANIN (Count of Base Classes) metric 210
Ignore category, C++ Includes 72
Ignore directories in include names field 72
Ignore Parent Overrides field 46
Implementation fields field, JOVIAL 85
Implemented By menu, graphical views 248
Implements menu, graphical views 247
implicit special member functions, C++ 70
Implicitly Declared Objects report 207
import files
 adding to project, Python 91
Import report 204
imported entities, displaying in graphical views 248
importing tool commands 294
Imports category
 Python 91
Imports menu, graphical views 248
-importusertools option, understand command 305
inactive code, cross-reference information for, C++ 70
inactive lines, hiding 163
Include By View 239
Include File Cross-Reference report 200
include files
 adding as source files, assembly 65
 adding as source files, C++ 71, 78
 adding as source files, FORTRAN 82
 adding as source files, PL/M 90
 adding before each project file, C++ 72
 adding in bulk, assembly 66, 68
 adding in bulk, C++ 71, 78
 adding in bulk, FORTRAN 82
 adding in bulk, PL/M 90
 adding to project, assembly 65
 adding to project, C++ 71, 77
 adding to project, COBOL 67
 adding to project, FORTRAN 82
 adding to project, PL/M 90
 compiler path for, C++ 69
 displayed in graphical views 248

environment variables in paths for, assembly 66, 68
environment variables in paths for, C++ 71, 78
hierarchy of, for source files 237, 239
ignoring during analysis, C++ 72
overriding MSVC project settings for 58
Pascal search path 89
replacement text for, C++ 73
replacement text for, FORTRAN 82
replacement text for, PL/M 90
reports about 200
system include files, C++ 72
Include line numbers in rich text field 108
Include Paths button 116
Include View 237, 239
IncludeBy View 237
Included By menu, graphical views 248
Includes category
 C++ 71, 77
 FORTRAN 82
 PL/M 90
Includes menu, graphical views 248
Indent field, Editor options 106
indentation 108, 109
 fixing 164
Information Browser (IB) 25, 28, 125
 architecture information in 177
 copying information in 125, 129
 displaying from entity in Entity Filter 125
 displaying source of selected entity 128
 entity information displayed by, choosing 126
 expanding and collapsing the tree 126
 for current entity 156
 for current file 156
 history for 126, 130
 location of 20
 metrics in 129, 214
 multiple occurrences open 127
 References in 128
 right-clicking in 24
 saving information in 125, 129
 searching 127
 synchronizing 127
Information Browser option, View menu 125
Inherited By menu, graphical views 248
Inherits menu, graphical views 248
initialization files 22
Insert Spaces Instead of Tabs field, Editor options 106

instance methods 210
instance variables 210
Instant Search 141
Instant Search menu option 141
Instantiated From View 237
instantiation
 of compilation units 237
 of generic types 237
 of generic units 237
 reports about 204
Instantiations View 237
interactivity during background processing 95
interfaces
 listing in Filter Area 25
 reports about 199
Internal Dependencies graph 179
interrupt handlers, listed as unused program units 208
intrinsic functions, parsing, FORTRAN 81
Intrinsic menu, graphical views 248
Intrinsics file field, FORTRAN 81
Invocation Tree report 203
Invocation View 237
invocations
 reports about 203
 views of 237, 238, 240, 242
Invocations menu, graphical views 249

J

.jar files 83
Java
 configuration for 83
 versions supported 15
Java category, Project Configuration dialog 83
Javascript files 15
JDK, versions supported 15
JNI external entities, Java 83
JOVIAL
 configuration for 85
 directories for !COPY directives 86
 versions supported 15
JOVIAL category, Project Configuration dialog 85
JPEG format, saving graphical views as 255
Jump to Matching Brace menu option 163
Jump to Matching Directive menu option 163

K

Key Bindings category, Understand Options dialog 100, 165
Key Bindings option, Help menu 16, 165
keyboard mappings 100
Keyboard Scheme field, Key Bindings options 100
keywords, blue text for 110
KNI external entities, Java 83

L

Language auto-architecture 30, 177
Languages category, Project Configuration dialog 41
Last Parse Log option, View menu 115
-lastproject option, understand command 305
-lastproject_cwd option, understand command 305
Layout menu, graphical views 249
layout, for graphical views 249
LCOM (Percent Lack of Cohesion) metric 210
Less memory usages versus speed field, Ada 62
Level menu, graphical views 250
libraries, standard
 Ada, directory for 62
 including in parse log 101
 Pascal, paths for 88
Library directories field, Ada 63
license server
 connection timeout 305
licensing
 specifying level from Help menu 14
Licensing option, Help menu 14
line endings for source files 105
line numbers, displaying in Source Editor 158
line termination style
 for reports 193
 for saving source files 106
Lines metric 212
@lisfile.txt file 304
Local menu, graphical views 250
local object declarations, including in database, C++ 70
local parameters, listed in Entity Locator 26
Lucene, Apache syntax 141

M

Macintosh line termination style 106
Macro Cross-Reference report 200
macros

- adding in bulk, C++ 74
- changing, Ada 64
- compiler-specific, C++ 69
- defining 118
- defining in command line 304
- defining on command line, Ada 64
- defining, Ada 63
- defining, C++ 73, 79
- of editing changes, recording and replaying 165
- expansion text for, C++ 70
- importing, Ada 64
- listing in Filter Area 25
- as octagons, in graphical views 245
- overriding MSVC project settings for 58
- recording references when expanding, C++ 70
- reports about 199, 200
- undefined, C++ 75
- See also* objects
- Macros category
 - Ada 63
 - C++ 73, 79
 - FORTTRAN 82
 - Pascal 88
- Main subprograms field, Ada 63
- makefile 306
- Manage Architectures option, Project menu 178, 184
- Margins field, Editor options 107
- "Mastering Regular Expressions" (O'Reilly) 152
- Max Inheritance Tree metric 210
- McCabe (Cyclomatic) complexity 205, 212
- members
 - cohesion of 210
 - default 246
 - displayed in graphical views 250
 - graphical views of 240
 - private 251
 - protected 251
 - public 251
- Members menu, graphical views 250
- memory
 - caching include files, C++ 70
 - optimizing analysis to use less, Ada 62
- menu bar 20
- menus. *See specific menus*
- merging, in Entity Comparison area 281
- methods
 - cohesion of 210
 - instance 210
 - listing in Filter Area 25
 - metrics about 210
- metrics 178, 183
 - calculating automatically 51
 - configuring 52
 - displayed in Information Browser 129
 - exporting to CSV file 51, 214, 218
 - exporting to HTML 51, 214, 217
 - for project 215, 216
 - graphs for 220
 - in Information Browser 214
 - list of, online 205, 209, 214, 215
 - reports about 209, 210, 211, 212, 214
 - selecting 53
- Metrics Browser 214
- Metrics category, Project Configuration dialog 52
- Metrics Export 178
- Metrics Export menu option 184
- Metrics menu 311
 - Browse Metric Charts option 224
 - Browse Metrics option 214, 216
 - Configure Metric Charts option 214, 220
 - Export Metrics option 51, 214, 218
 - Metrics Summary option 214, 215
- Metrics Summary 178, 183
- Metrics Summary option, Metrics menu 214, 215
- Microsoft Visio files, saving graphical views as 256
- Microsoft Visual C++ project. *See* MSVC project
- Microsoft Visual C++, as editor 112
- minus sign (-) collapsing tree in Information Browser 126
- Modified metric 212
- modules, listing in Filter Area 25
- MSVC project, importing into Understand project 37, 57
- multiple users, initialization files for 22

N

- Name menu, graphical views 250
- named root portability mode 47
- named roots 103
- Namespaces category, Pascal 88
- Navigation category, Editor options 111
- Navigation Mode, Source Editor 111
- nested comments, C++ 69
- New Architecture option, Project menu 185, 186
- New File option, File menu 165

-new option, understand command 305
 New Project option, File menu 35
 New Project Wizard 35, 102
 next button 18
 Next icon 29
 NIM (Count of Instance Methods) metric 210
 NIV (Count of Instance Variables) metric 210
 No Truncation text option 252
 No Wrap text option 253
 -no_splashscreen option, understand command 305
 NOC (Count of Derived Classes) metric 210
 -noproject option, understand command 305
 NTFS filesystem 95

O

Object Cross-Reference report 199
 Objective C/C++ 15
 configuration 76
 enabling 41
 object-oriented metrics 210
 objects
 displayed in graphical views 251
 listing in Project Window 25
 reports about 199, 207
 Objects menu, graphical views 251
 octagons in graphical views 245
 online help 16, 21
 Open File option, File menu 165
 Open last project at startup field, General options 94
 Open Project option, File menu 21
 Open Understand 1.4 Project option, File menu 119
 Operators menu, graphical views 251
 operators, displayed in graphical views 250, 251
 Options option, Tools menu 93
 Options subcategory, Reports 55
 or operators, including in strict complexity, Ada 62
 Output subcategory, Reports 54, 191

P

Package View 240
 packages
 declaration structure for 240
 listing in Filter Area 25
 reports about 204
 With By relationships for 240
 With relationships for 240

page guide, showing 106
 Page Setup option, File menu 174
 parallelograms, in graphical views 245
 parameters
 cross-reference information for, C++ 70
 relationships between formal and actual, Ada 62
 reports about 199, 207
 See *also* objects
 Parameters menu, graphical views 251
 parent
 of class. See base classes
 of entity 237
 Parent Declaration View 240, 242
 Parent Lib Unit View 237
 parentheses, matching. See brackets
 Parse Log window 115
 reopening previous Parse Log 153
 saving to text file 115
 parsing. See analyzing projects
 Pascal
 configuration for 87
 macro definitions 88
 namespaces directory 88
 search paths 89
 standard libraries, paths for 88
 versions supported 15
 Pascal category, Project Configuration dialog 87
 Patch File area, Change Results window 279, 282
 patching lines of context, in Entity Comparison area
 281, 282
 PATH definition 298
 path highlighting, in graphical views 234
 pattern matching. See regular expressions
 PC line termination style
 for reports 193
 for saving source files 106
 .pcn file 198
 PDF, saving graphical views to 258
 Percent Lack of Cohesion metric 210
 period (.) in regular expressions 151
 Perl
 API for custom reports 32
 CodeCheck scripts 272
 interface for custom reports 196, 214
 scripts 19
 Perl API Documentation option, Help menu 16, 32, 214
 permission checking 95

PHP files 15, 92
 pin icons. *See* pushpin icons
 pink background for text 110
 PL/M
 configuration for 89
 versions supported 15
 PL/M category, Project Configuration dialog 89
 plus sign (+)
 expanding tree in Information Browser 126
 in regular expressions 151
 PNG format, saving graphical views as 255
 Popup Menu category, Tool Configurations dialog 291
 Portability category, Understand Options dialog 103
 portability of project 47
 pound sign (#) comment in command line file 304
 pragma statements, defining macros referenced in, Ada 63
 Predeclared entities file field, Pascal 87
 Prefix Headers Category, C++ Includes 79
 Prepend the names of externally linkable entities with field
 C++ 70
 FORTRAN 81
 Prepend the names of JNI/KNI external entities with field, Java 83
 preprocessor directives, C++ 73
 Preprocessor field, Ada 61
 preprocessor macros. *See* macros
 preprocessor statements, green text for 110
 preprocessor support, enabling, FORTRAN 81
 previous button 18
 Previous icon 29
 Print Entity Graph option, File menu 257
 Print option, File menu 174
 printing
 CodeCheck results 269
 graphical views 257
 source files 174, 257
 Private Members menu, graphical views 251
 procedures
 in graphical views 251
 reports about 198
 See also invocation; functions; program units
 Program Unit Complexity report 205
 Program Unit Cross-Reference report 198
 Program Unit Metrics report 210
 program units
 metrics about 210
 reports about 202, 203, 204, 205, 208, 210
 With relationships for 204, 208
 programming languages
 selecting for project 41
 setting in Source Editor 160
 project 19
 adding source files on command line 300
 analyzing (parsing). *See* analyzing projects
 closing 22
 closing automatically when opening new project 98
 configuring 39
 converting from Understand 1.4 to 2.5 119
 creating 35
 creating, with New Project Wizard 102
 directory hierarchy for, displaying 130
 existing, opening 21
 metrics for 209, 215, 216
 opening most-recent project at startup 94
 portability of 47
 saving configuration of 40
 toolbar for 154
 Project Browser 130
 Project Browser option, View menu 130
 Project Configuration dialog 39, 191
 Ada category 61
 Assembly category 65
 C# category 79
 C++ (Strict) category 76
 C++ category 69
 COBOL category 67
 File Options category 49
 File Types category 48
 Files category 42
 FORTRAN category 80
 Java category 83
 JOVIAL category 85
 Languages category 41
 Metrics category 52
 Pascal category 87
 PL/M category 89
 Python category 91
 Reports category 54
 saving configuration 40
 Scheduled Activities category 50
 Visual Studio category 57
 Web category 92

-
- Project Graphs option, Graphs menu 32, 196
 - Project Interactive Reports option, Reports menu 32, 196
 - Project menu 310
 - Analyze All Files option 51, 114
 - Analyze Changed Files option 114
 - Browse Architectures option 177
 - Configure option 39
 - Manage Architectures option 184
 - New Architecture option 185, 186
 - Project Overview Charts option 229
 - Rescan Project Directories option 46, 50
 - Rescan Watched Project Directories option 46
 - Project Metrics report 209
 - Project Overview Charts option, Project menu 229
 - Prompt before closing the current project field, User Interface Alerts options 98
 - Prompt for missing includes field, C++ Includes 72, 78
 - Prompt on parse errors field
 - Ada 63
 - FORTRAN 81
 - \$Prompt variables 285, 286, 287
 - prompt when files modified externally 106
 - Protected Members menu, graphical views 251
 - Public Members menu, graphical views 251
 - pushpin icons 18
 - Python
 - API for custom reports 32
 - configuration for 91
 - versions supported 15
 - Python API Documentation option, Help menu 16, 32
 - Python category, Project Configuration dialog 91
- Q**
- quality reports 205
 - question mark (?)
 - in regular expressions 151
 - wild card 150
 - quiet_startup option, understand command 305
 - quote (") prefixing octal constants or string literals 81
 - quotes (""") surrounding normal includes 72
- R**
- read-only access 49
 - Recent files most recently use list field, User Interface Lists options 97
 - Recent projects most recently use list field, User Interface Lists options 97
 - Recent Projects option, File menu 21
 - Record Macro option, Tools menu 165
 - rectangles, in graphical views 245
 - rectangles, rounded, in graphical views 245
 - rectangular area, copy and paste 160
 - red project file icon 159
 - red text 110
 - Reference Count option 179
 - reference files, C# 79
 - References category, C# 79
 - References, in Information Browser 128
 - regular expressions
 - book about 152
 - in filters for Entity Locator 150
 - in Find dialog 155
 - Reindent Selection command 164
 - relationship 19
 - relative portability mode 47
 - Rename Architecture menu option 178
 - Rename Declaration View 241
 - rename declarations
 - displayed in graphical views 251
 - reports about 204
 - Renames menu, graphical views 251
 - Renames report 204
 - Replace in Files area 146
 - Replace in Files option, Search menu 146
 - Replacement Text category
 - C++ Includes 73
 - FORTRAN Includes 82
 - PL/M Includes 90
 - Replay Macro option, Tools menu 165
 - reports 32, 191
 - anti-virus software, turning off while generating reports 193
 - canceling generation of 193
 - categories of 195
 - colors in 193
 - configuring 191
 - cross-reference reports 197
 - customizing 32
 - customizing with Perl or C 196, 214
 - entity name format in 55
 - filenames for 194
 - generating 191, 193
-

- generation time of 55, 193
- HTML output 54, 193
- HTML output, entity index for 194
- list of, choosing from 56, 192
- metrics reports 209, 214
- quality reports 205
- structure reports 202
- text output 55, 193
- viewing 194
- See also specific reports*
- Reports category, Project Configuration dialog 54
 - Options subcategory 55
 - Output subcategory 54, 191
 - Selected subcategory 56, 192
- Reports menu 311
 - Configure Reports option 191
 - Generate Reports option 193
 - Project Interactive Reports option 32, 196
 - View Reports option 194
- Rescan project before parsing changed files field 101
- Rescan Project Directories option, Project menu 46, 50
- Rescan Watched Project Directories option 46
- Reuse window when requesting a similar type of graph
 - from within window field, General options 99
- RFC (Count of All Methods) metric 210
- Right-click Menu
 - external tools available in 285, 291
 - Find in Files option 143
- right-clicking 18, 121
 - + or - sign in Information Browser tree 126
 - on Analyze icon 114
 - anywhere in Understand 23
 - in Architecture Browser 178
 - in background of graphical views 245
 - bold heading in Information Browser 126
 - creating windows by 23
 - on entities in graphical views 233
 - on entities to display source 128
 - on entities in Entity Locator 26
 - on entities in Entity Locator entities 149
 - on entities in Information Browser 24
 - on entities in Source Editor 23, 29
 - in Entity Filter 24
 - in Entity Locator 148
 - on Entity Locator column headers 149, 150
 - in Information Browser 129
 - on Selector window 154

- reusing windows by 23, 121
 - on selected text 160, 164
 - in Source Editor 161
 - on Source Editor tabs 167
 - See also Ctrl+right-click*
- RO (read-only) indicator, in Source Editor 160
- Root filters, Entity Filter 124
- Routines menu, graphical views 251
- routines. *See* functions; procedures
- row colors, alternating 96
- Run Command option, Tools menu 295
- RW (read-write) indicator, in Source Editor 160

S

- Save all modified editor windows when application
 - loses focus field, General options 94
- Save All option, File menu 162
- Save comments associated with entities field
 - Ada 62
 - C 77
 - C++ 70
 - Java 83
- Save duplicate references field, C++ 70
- Save macro expansion text field
 - C++ 70
- Save on command field, User Interface Alerts options 98
- Save on parse field, User Interface Alerts options 98
- Save option, File menu 162
- saving edits automatically 94
- Scale menu, graphical views 252
- Scheduled Activities category, Project Configuration dialog 50
- Scheduler option, Tools menu 50, 51
- Scientific Toolworks website 15
- SciTools forum 196
- Scope Information, Contextual Information Sidebar 156
- Scope List option, View menu 159
- Scope List, Source Editor 159
 - See also* Structure Browser, Source Editor 156
 - toolbar for 154
- scripts
 - CodeCheck 272
 - Perl 19
- Search for include files among project files field, C++
 - Includes 72, 78
- Search menu 140, 309

- Find Entity option 148
- Find in Files option 27, 143
- Find option 155
- Go to Matching Brace option 163
- Instant Search option 141
- Replace in Files option 146
- Search Paths Category, Pascal 89
- searching
 - files 140, 143
 - graphical views 233
- Select All option, Edit menu 160
- Select Block menu option 163
- Selected subcategory, Reports 56, 192
- selecting text 280
- Selector area 153
- sharp sign (#) comment in command line file 304
- shortcut commands 100
- Show Edge Labels options 255
- Show Page Guide field, Editor options 106
- Show standard library files field, Analyze options 101
- Show tabs field, User Interface options 96
- Show the Getting Started dialog on startup field,
 - General options 94
- Show the Splash-Screen on startup field, General
 - options 94
- Simple Invocation Tree report 203
- Simple With Tree report 204
- SlickEdit editor 112
- sliding frames 18
- SlowConnect option, understand command 305
- Soft Wrap option, View menu 165
- "Software Engineering with Ada" (Booch) 241
- Sort menu, graphical views 252
- Sort Selection command 165
- sorting entities 123
- Sound beep field 98
- Sound beep on parse completion field, Analyze options
 - 101
- Source Editor 29, 158
 - auto-complete options 108
 - auto-indent options 108, 109
 - bookmarks, creating 166
 - bookmarks, navigating 166
 - bracket matching in 163
 - Browse Mode in 160
 - case, changing in 164
 - closing files 162
 - colors in, customizing 110, 158
 - colors in, default 110
 - commenting and uncommenting code 164
 - configuring 105
 - Contextual Information Sidebar (CIS) in 156
 - copying to clipboard from 160
 - creating files 165
 - displaying by right-clicking on entities 128
 - displaying from Find Results window 27, 145
 - external editor replacing 112
 - folding (hiding blocks in) code 163
 - hiding inactive lines in 163
 - history of locations visited in, moving through 153
 - jumping to specific line number 160
 - keystrokes in, list of 165
 - language, setting 160
 - line numbers displayed in 158
 - list of specific structures in 156, 159
 - location of 20
 - macros of editing changes, recording and replaying
 - 165
 - moving between windows of 29
 - opening files 165
 - printing from 107, 174
 - read-write (RW) and read-only (RO) indicators 160
 - right-clicking in 23, 29, 161
 - right-clicking tabs in 167
 - saving files 162
 - Scope List in 159
 - searching and replacing text in source files 155
 - searching in source files 140
 - status bar in 159
 - status icon in 159
 - tabs in, controlling behavior of 167
 - toolbar for 154
- source files
 - adding to project 37, 42, 131
 - adding to project on command line 300
 - analyzing using projects. *See* project
 - closing 152, 162
 - creating 165
 - declaration structure for 240, 243
 - deleting from project 45
 - directories for, adding to project 43
 - directories for, deleting from project 45
 - displaying by double-clicking entity 128
 - displaying by right-clicking on entities 128

- displaying from Find Results window 27, 145
- editing. *See* Source Editor
- encoding for 49, 105
- excluding from source list 44
- external editor for 112
- imported classes in 243
- imported, report about 204
- include files specified as, assembly 65
- include files specified as, C++ 71, 78
- include files specified as, FORTRAN 82
- include files specified as, PL/M 90
- include hierarchy for 237, 239
- line endings for 105
- list of, generating from command line 304
- listing in Filter Area 25
- metrics about 211
- moving between windows of 29
- opening 165
- opening from graphical view 233
- overriding settings for 46
 - as parallelograms, in graphical views 245
- portability of 47
- printing 174, 257
- printing, configuration for 107
- read-only access for 49
- removing from project 131
- saving 162
- searching 27, 131, 140, 143
- Sources tab, Project Configuration dialog 42
- sourcestyles.css file 193
- spaces, converting tabs to 106
- Spacing menu, graphical views 252
- special member functions, C++ 70
- splash screen, displaying at startup 94
- split
 - Source Editor 164
 - workspace 152
 - workspace, toolbar for 154
- Sql menu, graphical views 252
- SQL, embedded, in Pascal 87
- square brackets. *See* brackets
- src.jar file 83
- src.zip file 83
- Standard field, Ada 62
- standard libraries
 - Ada, directory for 62
 - displaying in parse log 101
 - Pascal, paths for 88
- Standard Library Paths category, Pascal 88
- standards, CodeCheck 262
- Start menu, Understand commands in 21
- startup
 - Getting Started dialog displayed on 94
 - Splash-Screen displayed on 94
- static functions, displayed in graphical views 252
- Static menu, graphical views 252
- status bar, Source Editor 159
- status icon, in Source Editor 159
- status line 20
- strict C/C++ analyzer 41, 76
- strict complexity, count and/or operators in, Ada 62
- Strict metric 212
- Structure Browser, Contextual Information Sidebar 156
- structure reports 202
- structure views 31, 233, 240
 - See also* declaration views; graphical views
- Styles category, Editor options 110
- subprograms, listing in Filter Area 25
- subtraction sign (-) collapsing tree in Information Browser 126
- Sum Complexity metrics graph 220
- support contact information 15
- SVG format, saving graphical views as 255
- switches, command line
 - understand command 305
- synchronizing Information Browser 127
- "System Design in Ada" (Buhr) 241
- system include files, C++ 72

T

- tab, automatic 108, 109
- Table View of graphs 230
- Table View, for metrics graphs 222
- tabs for windows, displaying 96
- tabs, converting to spaces 106
- Task View 241
- tasks
 - declaration structure for 241
 - With relationships for 241
- technical support contact information 15
- text
 - comparing 278
 - copying to clipboard 160
 - generating reports as 32, 55, 191, 193

- selecting 160
- size of, in graphical views 252
- viewing reports as 194
- Text Comparison window 278
- text favorite 138
- Text menu, graphical views 252
- TextPad editor 112
- title areas, filenames in 97
- title bar 18
- Title Formats field, User Interface options 97
- title page 55
- Toggle Overtyping option, Edit menu 165
- tool commands, importing and exporting 294
- Tool Configurations dialog
 - Popup Menu category 291
 - Toolbar category 293
- toolbar
 - external tools available from 293
 - hiding and displaying icons 154
 - location of 20
 - visibility of icons in, controlling 294
- Toolbar category, Tool Configurations dialog 293
- Tools menu 313
 - Compare Arbitrary Text option 278
 - Compare Entities option 277
 - Compare Files/Folders option 275
 - Editor Macros option 165
 - Options option 93
 - Run Command option 295
 - Scheduler option 50, 51
 - User Tools option 284, 285, 292, 294
- tools, external. *See* external tools
- tooltips 111
- Treat system includes as user includes field
 - C++ Includes 72
- Tree row indentation field, User Interface options 96
- treemap
 - CodeCheck 268
 - metrics 223
- Truncate column field
 - FORTRAN 81
 - JOVIAL 85
- Truncate Long text option 252
- Truncate Medium text option 252
- Truncate Short text option 252
- truncation at column
 - FORTRAN 81

- JOVIAL 85
- Turbo Pascal 87
- Type Cross-Reference report 199
- Type Derived From View 237
- Type Tree View 237
- types
 - displayed in graphical views 253
 - as hexagons, in graphical views 245
 - information about 240, 243
 - listing in Filter Area 25
 - reports about 199, 207
 - types derived from 237
- Types menu, graphical views 253
- Typetext menu, graphical views 253

U

- .udb file extension 19, 34
- UML Class Diagram 240, 242
- Uncomment Selection menu option 164
- und command 297, 298
 - adding files to project 300
 - analyzing a project 303
 - options in latest version, listing 300
- Undefined Macros button 118
- undefined macros, C++ 75
- Undefines category, C++ Macros 75
- Understand
 - compared to compiler 34
 - contact information 15
 - features of 13
 - multiple users for 22
 - online help for 16, 21
 - starting 21
 - starting from command line 305
 - windows in 18, 20
- Understand 1.4, converting to 2.5 119
- understand command 305
- Understand Options dialog 93
 - Analyze category 101
 - Command Window category 102
 - Configure category 102
 - Dependency category 104
 - Editor category 105
 - General category 94
 - Graphs category 113
 - Key Bindings category 100, 165
 - Portability category 103

User Interface category 96
 undocking windows 18
 Unicode file handling, in Entity Comparison area 281
 Uninitialized Items report 207
 UNIX
 line termination style, for reports 193
 line termination style, for saving source files 106
 unknown entities, displaying in graphical views 253
 Unknown menu, graphical views 253
 unresolved entities
 displaying in graphical views 253
 Unresolved menu, graphical views 253
 unresolved variables, listed in Entity Locator 26
 Unused Object report 207
 Unused Program Unit report 208
 Unused Type report 207
 Unused Variables and Parameters report 207
 Update Information Browser field, Browse Mode Editor
 options 111
 uperl command 19
 Use alternating row colors field, User Interface options
 96
 Use case-insensitive lookup for includes field
 C++ Includes 72
 Use default working directory field, General options 95
 Use include cache field, C++ 70
 Use preprocessor field, FORTRAN 81
 use statements, reports about 208
 Use the New Project Wizard when creating new
 projects field, Configure options 102
 Used By View 237
 Usedby Menu, graphical views 253
 User Interface category, Understand Options dialog 96
 user interface, parts of 20
 User Tools option, Tools menu 284, 285, 292, 294
 users, multiple, initialization files for 22
 Uses Menu, graphical views 253
 Uses Not Needed report 208
 Uses View 237

V

variables
 displayed in graphical views 253
 instance 210
 listed in Entity Locator 26
 metrics about 210
 reports about 199, 207
 as rounded rectangles, in graphical views 245
 uninitialized, report about 207
 unresolved 26
 See also objects
 Variables menu, graphical views 253
 .vcp file extension 57
 .vcw file extension 57
 Version field
 Ada 61
 FORTRAN 80
 Java 83
 JOVIAL 85
 Pascal 87
 Python 91
 vertical bar (|) in regular expressions 151
 Vertical Non-Crossing layout option 250
 VHDL
 terminology 91
 versions supported 15
 vi editor 112
 View menu 310
 Bookmarks option 154, 166
 Browse Mode option 160
 Contextual Information option 156
 Entity Locator option 26, 148
 Favorites option 27, 136
 Fold All option 163
 Hide Inactive Lines option 163
 Information Browser 125
 Last Parse Log option 115
 Project Browser option 130
 Scope List option 159
 Soft Wrap option 165
 Window Selector option 153
 Zoom option 158, 167
 View Reports option, Reports menu 194
 View SciTools Blog option, Help menu 16
 views. *See* declaration views; graphical views;
 hierarchical views
 violations
 checking for 262
 ignoring 270
 Visio files, saving graphical views as 256
 -visit option, understand command 306
 Visit Source field, Browse Mode Editor options 111
 Visual C++, as editor 112

Visual Studio
 as external editor 112
 files, synchronizing with Understand project 37, 57
Visual Studio category, Project Configuration dialog 57

W

-wait option, understand command 306
warnings, displaying from Parse Log window 115
watched directories
 scanning 46, 50
 setting 44, 45
Web category, Project Configuration dialog 92
Web support 15
websites
 external editor information 112
 metrics, list of 205, 209, 214, 215
 O'Reilly and Associates 152
 Scientific Toolworks 15
 SciTools forum 196
white file icon 159
whitespace
 in Entity Comparison area 281
 indicators for 106
Whitespace field, Editor options 106
wild cards, in filters for Entity Locator 150
%WINDIR% environment variable 22
Window menu 152, 314
 Close All Document Windows option 152, 162
 Close option 152, 162
 Entity Graph option 233
 Windows option 153
Window Selector option, View menu 153
windows 18, 20
 animated opening and closing of 96
 closing 18
 creating with Ctrl+right-click 23, 121
 docking and undocking 18
 docking layout for 97
 filenames in title area of 97
 frames in, sliding 18
 graphs in, reusing 99
 list of 122
 list of open windows 153
 organizing 152
 reusing for graphical views 236
 reusing with right-click 23, 121

 tabs for, displaying 96
Windows category, User Interface options 99
Windows CE project 57
Windows CE workspace 57
Windows folder 22
Windows line termination style
 for reports 193
 for saving source files 106
Windows option, Window menu 153
With Bys menu, graphical views 253
With Tree report 204
With View 237
WithBy View 237
Withs menu, graphical views 253
Withs Not Needed report 208
WMC (Count of Methods) metric 210
workbench file 57
working directory, default 95
workspace file 57
Wrap Long text option 253
Wrap Medium text option 253
Wrap Mode field, Advanced Editor options 108
Wrap Short text option 253
wrapping lines
 display 165
 printing 108

X

XML Export 178, 189
XML output 225, 226
XML, sharing architectures using 189

Y

yellow background for text 111
yellow project file icon 159

Z

Zoom option, View menu 158, 167
zooming
 graphical views 234
 source views 158