



Understand™  
by SciTools

# *User Guide and Reference Manual*

**January 2022**



Understand is developed using a rapid iterative process with weekly releases. As such, you may discover a feature that has out-paced its documentation. Our support team will be happy to answer any questions about it while we work on the documentation, contact us at [support@scitools.com](mailto:support@scitools.com).

Scientific Toolworks, Inc.  
328 N Old Hwy 91, Ste A  
Hurricane, UT 84737

Copyright © 2022 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., 328 N Old Hwy 91, Ste A, Hurricane, UT 84737.

**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.

Reference Build: Understand 6.1 Build 1091 (1/22)

# Contents

## Chapter 1

### Introduction

What is Understand? .....	12
Languages Supported .....	13
For Those Who Don't Like to Read Manuals .....	14

## Chapter 2

### Parts and Terminology

Using Understand Windows .....	16
Understand Terminology .....	17
Parts .....	17
Starting Understand .....	19
Other Ways to Run Understand .....	20
Context Menus Are Everywhere .....	21
Quickly Find Things in Your Source .....	23
Entity Filter .....	23
Entity Locator .....	24
Instant Search .....	24
Find in Files .....	25
Favorites .....	25
Information Browser .....	26
Source Editor .....	27
Architecture Browser .....	28
Graphical Views .....	29
APIs for Custom Reporting .....	30

## Chapter 3

### Configuring Your Project

About Understand Projects .....	32
Project Storage .....	32
Creating a New Project .....	34
Using Buildspy to Create Projects .....	36
Creating Projects from Cmake Projects .....	36
Creating Projects from Visual Studio Projects .....	37
Creating Projects from Apple Xcode Projects .....	38
Project Configuration Dialog .....	39
Languages Category .....	41
Files Category .....	42
Adding Directories .....	43
Adding Files .....	44
Removing Directories and Files .....	44
Setting Overrides .....	45
File Type Options .....	47
File Options .....	48

Imports Options .....	49
History Options .....	51
Portability Options.....	52
Ada Options .....	54
Ada > Macros Category.....	56
Assembly Options.....	58
COBOL Options .....	60
COBOL > Copybooks Category.....	61
C++ Options .....	62
Strict C/C++ Mode Options .....	63
Fuzzy C/C++ Mode Options.....	66
C++ > Includes Category.....	68
C++ > Includes > Automatic Includes Category.....	70
C++ > Includes > System Includes Category.....	70
C++ > Includes > Frameworks Category .....	70
C++ > Includes > Ignore Category.....	70
C++ > Includes > Replacement Text .....	71
C++ > Macros Category.....	71
C++ > Macros > Undefines Category.....	72
C# Options .....	73
Fortran Options.....	74
Fortran>Includes Category.....	76
Other Fortran Categories .....	76
Java Options.....	77
Java > Class Path Category.....	78
JOVIAL Options .....	79
Jovial > !Copy Category.....	80
Pascal Options .....	81
Pascal > Macros Category.....	82
Pascal > Standard Library Paths Category .....	82
Pascal > Search Paths Category.....	82
PL/M Options .....	83
PL/M>Includes Category .....	83
Python Options .....	84
Python > Imports Category .....	85
VHDL Options.....	85
Visual Basic (.NET) Options .....	86
Web Options.....	86
Analyzing the Code.....	88
Improving the Analysis.....	90
Using the Missing Header Files Tool .....	90
Using the Undefined Macros Tool .....	91

**Chapter 4****Setting Options**

Understand Options Dialog .....	93
General Category .....	94
User Interface Category .....	96
User Interface > Lists Category .....	97
User Interface > Alerts Category .....	98
User Interface > Windows Category .....	99
User Interface > Application Styles Category.....	100
Key Bindings Category .....	101
CodeCheck Category .....	103
Analyze Category .....	103
Dependency Category .....	104
Editor Category.....	105
Editor > Advanced Category .....	107
Editor > Macros Category .....	110
Editor > Styles Category .....	111
Editor > Navigation Category.....	112
Editor > External Editor Category.....	113
Graphs Category .....	114
Annotations Category.....	118
User Tools Category.....	120
Python Category.....	120

**Chapter 5****Exploring Your Codebase**

PLEASE RIGHT-CLICK .....	122
Various Windows Explained... .....	123
Entity Filter .....	124
Using the Filter Field .....	124
Customizing the Display.....	125
Root Entity Types.....	125
Information Browser .....	126
Drilling Down a Relationship .....	127
Displaying More or Less Information .....	128
Searching the Information Browser .....	128
Syncing the Information Browser.....	129
Visiting Source Code .....	129
Visiting References .....	130
Viewing Metrics .....	130
Saving and Printing Information Browser Text.....	131
Entity History .....	131
Project Browser .....	132
Exploring a Hierarchy .....	134
Dependency Browser .....	135
What are Dependencies?.....	137

Exporting Dependencies .....	139
Exporting Dependencies to a CSV File .....	140
Exporting Dependencies to a CSV Matrix File.....	141
Exporting Dependencies to Cytoscape .....	141
Favorites.....	142
Creating a Favorite Entity.....	142
Creating a Favorite View .....	143
Using a Favorites Group .....	143
Creating a Plain Text Favorite .....	145
<b>Chapter 6</b>	<b>Searching Your Source</b>
Searching: An Overview .....	147
Instant Search.....	148
Find in Files .....	150
Search Results.....	152
Replace in Files .....	153
Entity Locator .....	155
Resizing Columns .....	155
Long versus Short Names .....	155
Column Headers .....	156
Choosing Columns.....	156
Filtering the List .....	157
Finding Windows .....	159
Source Visiting History.....	160
View Menu Commands .....	161
Displaying Toolbars .....	162
Session Browser .....	162
Searching in a File .....	163
Find Next and Previous .....	163
Find & Replace.....	163
Contextual Information.....	164
<b>Chapter 7</b>	<b>Editing Your Source</b>
Source Editor .....	166
File Icons .....	166
Scope List .....	167
Toolbar .....	168
Status Line .....	168
Selecting and Copying Text.....	169
Browse Mode .....	169
Context Menu.....	170
Hover Text .....	171
Saving Source Code.....	171
Refactoring Tools .....	172

Renaming Entities .....	173
Inlining Functions .....	174
Extracting Functions.....	175
Inline Temp .....	176
Extract Temp .....	178
Other Editing Features .....	179
Previewer .....	179
Bracket Matching .....	180
Folding and Hiding .....	180
Changing the Source Code Font Size .....	180
Splitting the Editor Window .....	181
Changing Case .....	181
Commenting and Uncommenting.....	182
Indentation .....	182
Line Wrapping .....	182
Insert and Overtype Modes .....	182
Sorting Lines Alphabetically.....	182
Keyboard Commands .....	182
Recording, Playing, and Saving Macros .....	183
Creating and Opening Files .....	183
Bookmarking .....	184
Managing Source Editor Tabs .....	185
Annotations.....	186
Viewing Annotations.....	187
Sharing Annotations.....	187
Searching for Annotations .....	187
Adding Annotations .....	188
Editing Annotations .....	188
Tagging Annotations .....	189
Attaching Files .....	189
Deleting Annotations .....	190
Managing Orphan Annotations .....	190
Printing Source Views.....	191

## Chapter 8

### Architecting Your Codebase

About Architectures .....	193
Browsing Architectures .....	194
Enabling Built-In Architectures.....	195
Context Menus for Architectures .....	195
Creating and Editing Custom Architectures .....	197
Creating Architecture Nodes .....	197
Editing Architecture Nodes.....	197
Adding Files and Entities to Nodes .....	198
Complete Coverage for Architectures .....	198

Managing Orphan Architecture Nodes . . . . .	199
Using the Architecture Designer . . . . .	199
Sharing Architectures . . . . .	201
Viewing Architecture Graphs . . . . .	201
Dependency Graphs . . . . .	201
Architecture Structure Views . . . . .	202
Checking Dependencies . . . . .	203
Viewing Architecture Metrics . . . . .	204

## Chapter 9

### Using Metrics

About Metrics . . . . .	206
Project Overview . . . . .	207
Metrics Browser . . . . .	208
Metric Definitions . . . . .	209
Exporting Metrics . . . . .	210
Metrics in the Information Browser . . . . .	212
Metrics Treemap . . . . .	213

## Chapter 10

### Using Graphical Views

Opening Graphs . . . . .	217
Graph Variants . . . . .	220
Controlling Graphical Views . . . . .	222
Graph Toolbar . . . . .	222
Graph Options . . . . .	224
Mouse Controls . . . . .	224
Classic Context Menu Controls . . . . .	225
Customization Panel Controls . . . . .	227
Context Menus for Graphs . . . . .	229
Saving Graphical Views . . . . .	236
Saving Views to Files . . . . .	236
Saving Views as Visio Files . . . . .	236
Saving Views as DOT Files . . . . .	237
Printing Graphical Views . . . . .	238
Graphical View Printing . . . . .	238
Importing Graphical View Plugins . . . . .	239

## Chapter 11

### Using CodeCheck for Standards Verification

About CodeCheck . . . . .	241
Running a CodeCheck . . . . .	242
Configuring Checks . . . . .	243
Selecting Files to Check . . . . .	245
Viewing Results . . . . .	246
Viewing Results by File . . . . .	246
Viewing Results by Checks . . . . .	247

	Viewing Results in the Locator.....	248
	Viewing Results by Treemap .....	249
	Viewing the Results Log.....	250
	Ignoring or Excluding Violations .....	250
	Ignoring Violations .....	251
	Adding Notes About Ignored Violations .....	252
	Using Baseline Ignore Settings .....	253
	Adding Automatic Ignores to Code .....	253
	Exporting CodeCheck Results .....	254
	Writing CodeCheck Scripts.....	255
	Installing Custom Scripts .....	257
<b>Chapter 12</b>	<b>Comparing Source Code</b>	
	Comparing Files and Folders .....	259
	Comparing Entities .....	262
	Comparing Text .....	263
	Comparing Projects .....	263
	Comparison Graphs .....	266
	Exploring Git History.....	267
	Exploring Differences .....	269
<b>Chapter 13</b>	<b>Running Tools and External Commands</b>	
	Configuring Tools .....	272
	Variables .....	274
	Adding Tools to the Context Menus .....	279
	Adding Tools to the Tools Menu.....	280
	Adding Tools to the Toolbar .....	281
	Importing and Exporting Tool Commands .....	282
	Running External Commands.....	283
	Running Plugins: Graphs and Interactive Reports .....	284
	APIs .....	284
	Interactive Reports.....	284
	Plugin Graphs .....	285
<b>Chapter 14</b>	<b>Command Line Processing</b>	
	Using the und Command Line .....	287
	Getting Help on Und.....	289
	Creating a New Project .....	289
	Converting a Legacy Project .....	289
	Adding Files to a Project .....	289
	Removing Items from a Project .....	291
	Getting Information about a Project and the License .....	291
	Modifying Project Settings .....	292
	Importing into a Project .....	292

Exporting from a Project .....	292
Analyzing a Project .....	293
Generating Reports .....	293
Generating Metrics .....	293
Using CodeCheck .....	294
Running Perl Scripts .....	294
Creating a List of Files .....	294
Using the understand Command Line .....	295
Using Buildspy to Build Understand Projects .....	296

## **Chapter 15**

### **Quick Reference**

File Menu .....	298
Edit Menu .....	299
Search Menu .....	299
View Menu .....	300
Project Menu .....	300
Architectures Menu .....	301
Metrics Menu .....	301
Graphs Menu .....	301
Checks Menu .....	302
Annotations Menu .....	302
Tools Menu .....	302
Compare Menu .....	303
Window Menu .....	303
Help Menu .....	304

---

## 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:

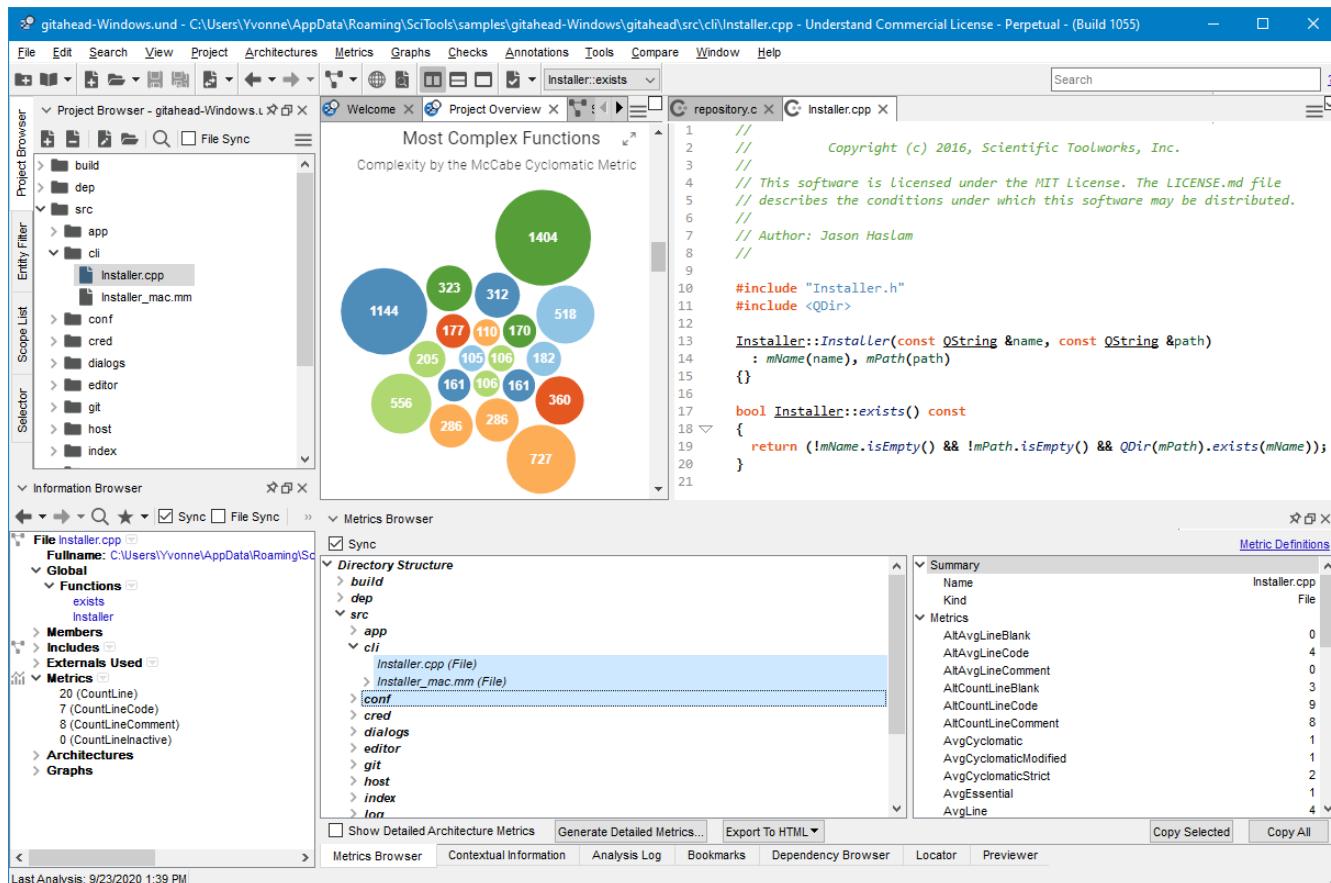
<b>Section</b>	<b>Page</b>
What is Understand?	12
Languages Supported	13
For Those Who Don't Like to Read Manuals	14

## What is Understand?

*Understand* is a static analysis tool focused on source code comprehension, metrics, and standards testing. 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, Assembly, Visual Basic, COBOL, Fortran, Java, JOVIAL, Pascal/Delphi, PL/M, Python, VHDL, and Web (PHP, HTML, CSS, JavaScript, Typescript, 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 has it been 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.

## Languages Supported

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

- **Ada:** *Understand* supports Ada83, Ada95, Ada05, and Ada2012 code, separately or in combination.
- **Assembly:** *Understand* supports assembly code for NXP Coldfire 68K, JIPSE MIL-STD-1750A, and IBM System/370.
- **Visual Basic:** *Understand* supports Visual Basic 2002 through 2015.
- **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. C++11, C++14, and C++17 are supported.
- **Objective C/Objective C++:** *Understand* supports Objective C and Objective C++ using the C/C++ parser.
- **C#:** *Understand* supports C# up to and including version 9.
- **COBOL:** *Understand* supports COBOL 85.
- **Fortran:** *Understand* supports FORTRAN 77, Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 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](mailto:support@scitools.com).
- **Java:** *Understand* supports Java through Java 13 and some of Java 14.
- **JOVIAL:** JOVIAL73 and JOVIAL3 are supported.
- **Delphi/Pascal:** *Understand* supports all versions of Embarcadero's Delphi language and Embarcadero's Turbo Pascal language. It also supports ISO 7185: 1990 (also known as Unextended Pascal) with HP 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 and Python 3.
- **VHDL:** Versions VHDL-87, VHDL-93, and VHDL-2001 are supported.
- **Web:** HTML, PHP, CSS, JavaScript, Typescript, and XML files are supported.

## For Those Who Don't Like to Read Manuals

If you are like many engineers at SciTools, 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:

- Click on one of the example projects in the welcome screen to download and open the project. You can also download projects using the **Help > Example Projects** menu. Open views using the View menu and play with *Understand*'s features.
- Read the tips provided within *Understand*. If you have disabled hints and would like to see them again, choose **Help > Reset All Hints**.
- Choose **Help > Help Content** from the menus.
- Visit [support.scitools.com](http://support.scitools.com) and read knowledge base topics that explain useful features.
- Visit [blog.scitools.com](http://blog.scitools.com) and read posts about ways people use *Understand*.
- Visit our YouTube channel to watch videos about using *Understand*.
- Subscribe to our newsletter to learn about ways our customers use *Understand*.
- If you have a specific question, click the **Chat** icon in the *Understand* Welcome tab and send us your question.



For information about installing *Understand*, see the [Installation category](#) on the support site.

For information about licensing, including using Offline mode, choose **Help > Licensing** and see the [Licensing category](#).

For information about privacy and to enable or disable statistical usage reporting, choose **Help > Privacy**.

For more advanced users, try these information sources:

- Choose **Help > About Understand** to see which build you are currently running.
- Visit [support.scitools.com](http://support.scitools.com) and read the Build Notes to see what functionality has been added recently.
- Choose **Help > Key Bindings** for extensive keystroke help.
- Choose **Help > Perl API Documentation** and **Help > Python API Documentation** for help on scripting. Java API documentation is provided in the doc/manuals/java subdirectory of the *Understand* installation.
- See the “API/Plugins” and “Power User Tips” categories on the [support site](#).

---

## Chapter 2      Parts and Terminology

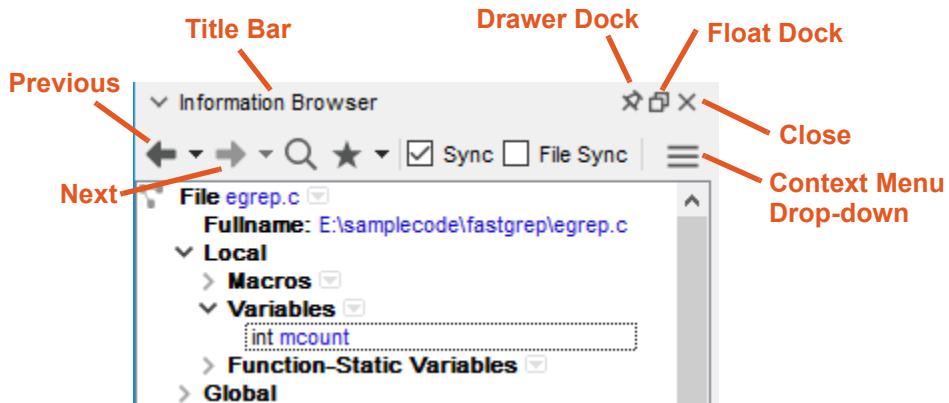
This chapter helps you put *Understand* to good use quickly and easily by describing the basic windows in *Understand*.

This chapter contains the following sections:

Section	Page
Using Understand Windows	16
Understand Terminology	17
Starting Understand	19
Context Menus Are Everywhere	21
Quickly Find Things in Your Source	23
Information Browser	26
Source Editor	27
Architecture Browser	28
Graphical Views	29
APIs for Custom Reporting	30

## 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 again to “pin” a drawer open.
- **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 *Understand* 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* opens a project using the most recent session layout. Changes to which tools, toolbars, tabs, and other features are open and their sizes and locations are saved automatically as part of the session. You can create and save multiple session layouts using the Session Browser (page 162).

## 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 (that is, an architecture's flattened expansion need not reference every source entity in the project), nor unique (that is, an architecture's flattened expansion need not maintain the set property).

**Database:** The database contains the results of the source code analysis in a format that can be rapidly searched. The database is the parse.udb" file, which is stored in the "local" subdirectory of the project folder. This database is regenerated as needed for each project. In previous versions of *Understand*, the database also contained all project setting information, which made it difficult to share projects. See *Project Storage* on page 32 for details.

**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.

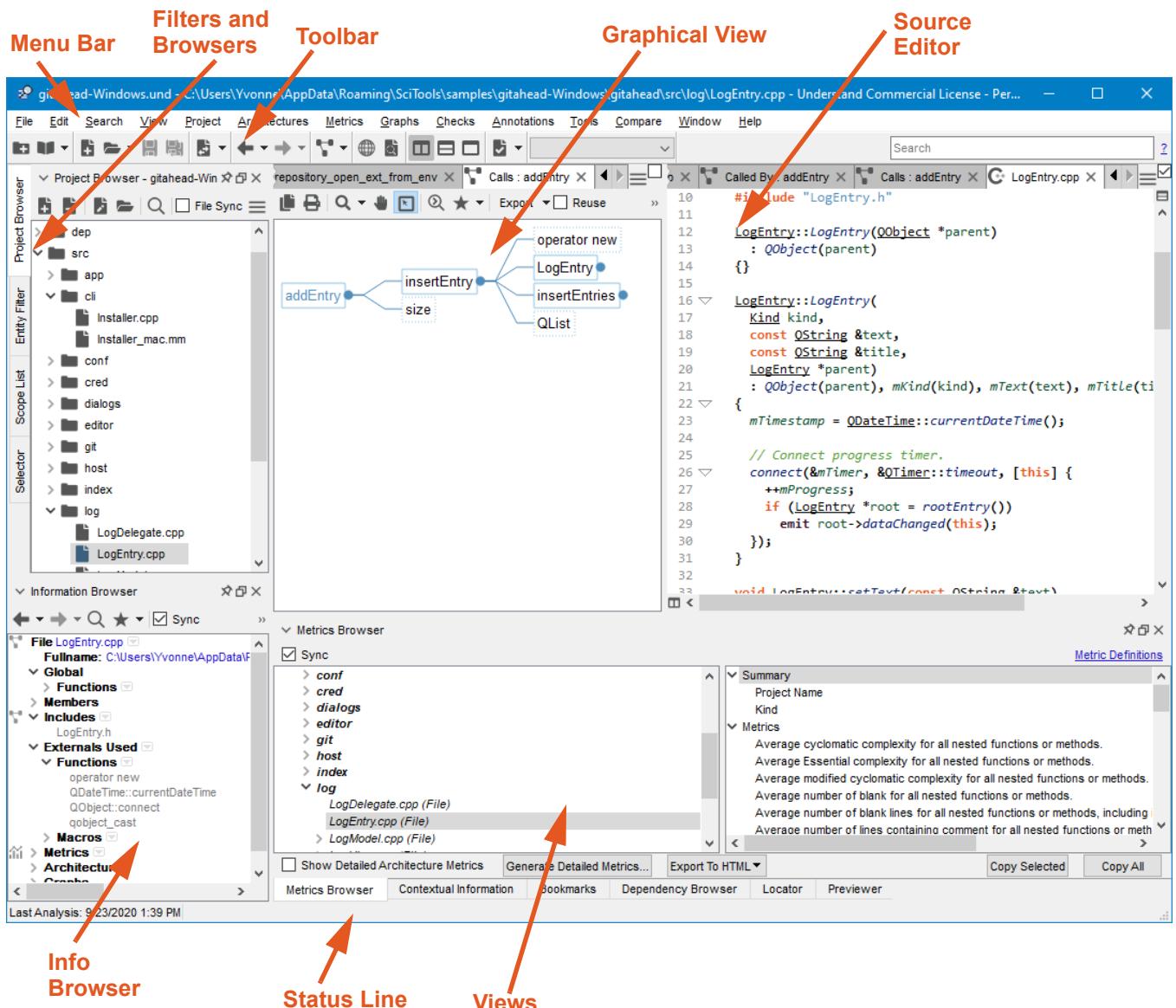
**Project:** The set of source code you have analyzed and the settings and parameters chosen. An .und project directory contains everything needed to share your project with others on your team.

**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 Python or Perl script. These can be run from within *Understand's* GUI, or externally via the "upython" or "uperl" command. The *Understand* Python and Perl APIs provide easy and direct access to all information stored in an Understand project.

### Parts

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



## Starting Understand

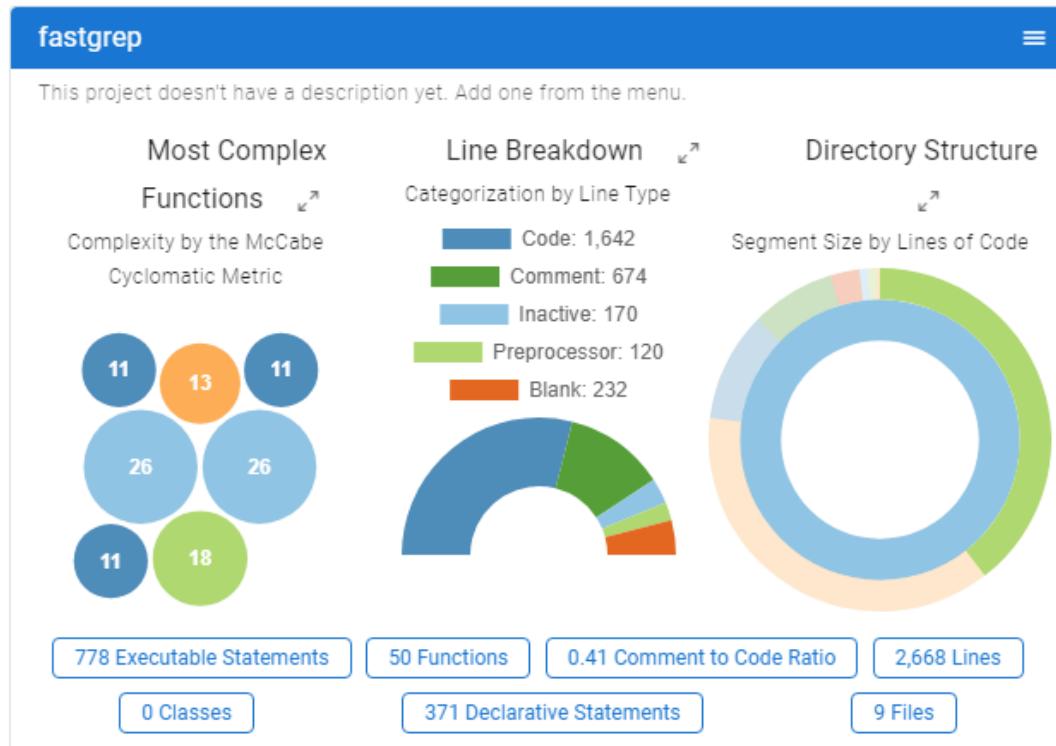
When you install *Understand* on Windows, a command to launch the software is added to your Windows **Start** menu.

When you start *Understand*, you see the **Welcome** page in the *Understand* window. To begin creating a new project, click **New** and see *Creating a New Project* on page 34 for details.



The Welcome page shows summary graphics—Line type breakdown, Directory structure, and Most complex functions—for sample projects available for download and any projects you have opened recently. Sample projects are provided that use C, C++, Ada, Delphi, VHDL, Visual Basic, Fortran, Python, C#, and Java.

Click any project's title bar to download and open it. If an existing project you want to open isn't listed, click **Open** and browse for it.



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

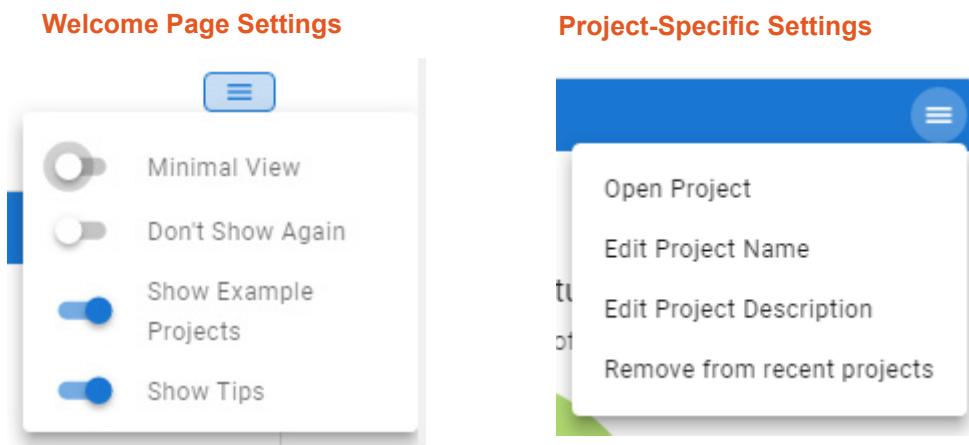
If you want to open a project that was created with an earlier version of *Understand* and convert it to the new project storage format, choose **File > Open > Legacy .udb Project**. Browse to select the file with a .udb extension that was used by earlier versions to contain projects instead of a directory. You will be asked whether you want

the new project to be stored in the same location as the source code or in a different location. Choosing the same location as the source code makes the project easier to share with others.

You can close the current project by choosing **File > Close <project\_name>.und**. Choose **File > Exit** to close *Understand*.

You can open a single project at a time in *Understand*. When you open a project, you are asked if it is OK to close the current project if one was open. Any changes you have made to the project settings and the arrangement of views and tools are saved automatically. If you have changed source code, you are prompted to save or discard the changes for each file individually. To learn about where the project files are saved and how you can store them in a source-code control system, see *Project Storage* on page 32.

To control the Welcome page display, use the drop-down menus for the overall page and for individual projects.



The Minimal View displays the projects on smaller cards with one summary graphic at a time.

Editing the project name and the project description on the Welcome page affects only what is shown on the Welcome page; it does not change the project itself.

If you have closed the Welcome page and want to reopen it, choose **Help > Show Welcome Page** from the menus. If you don't want to see the Welcome page every time you run *Understand*, toggle on the **Don't Show Again** option.

---

## 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 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:

- **Windows:** C:\Users\<Username>\AppData\Roaming\SciTools\Understand.ini
- **Linux/Unix:** ~/.config/SciTools/Understand.conf
- **MacOS:** ~/Library/Preferences/com.scitools.Understand.plist

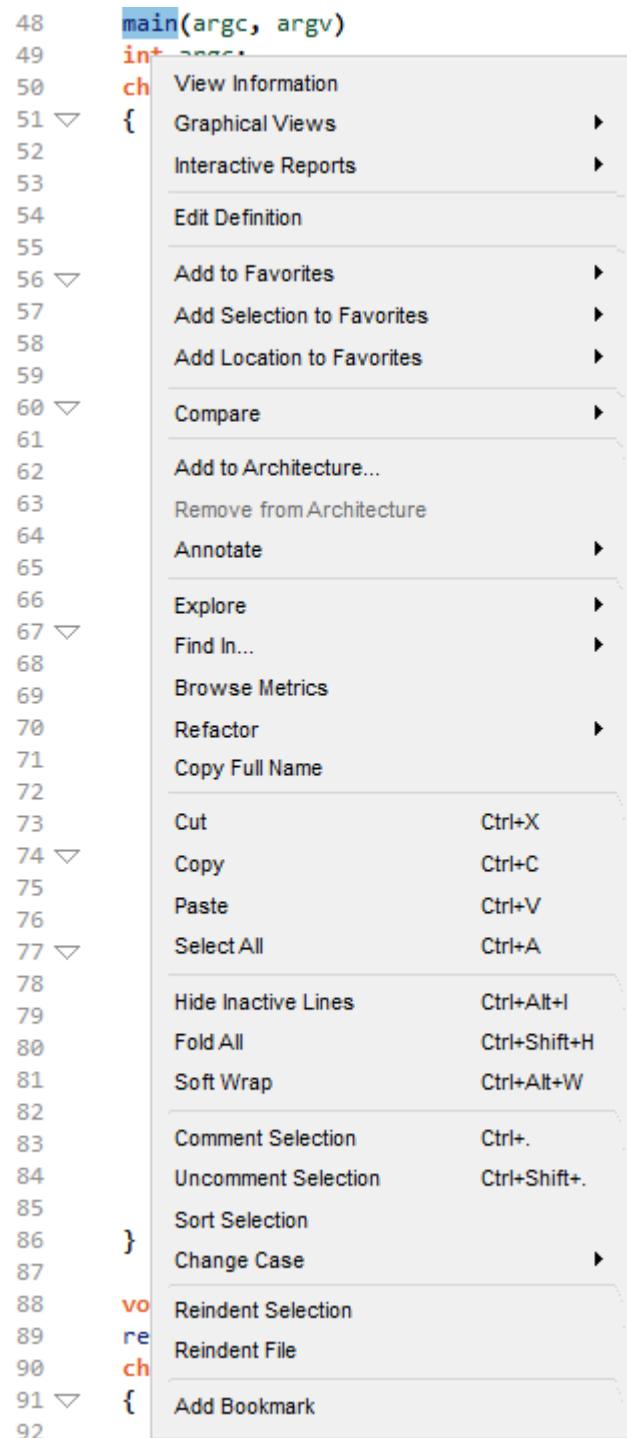
## Context Menus Are Everywhere

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.

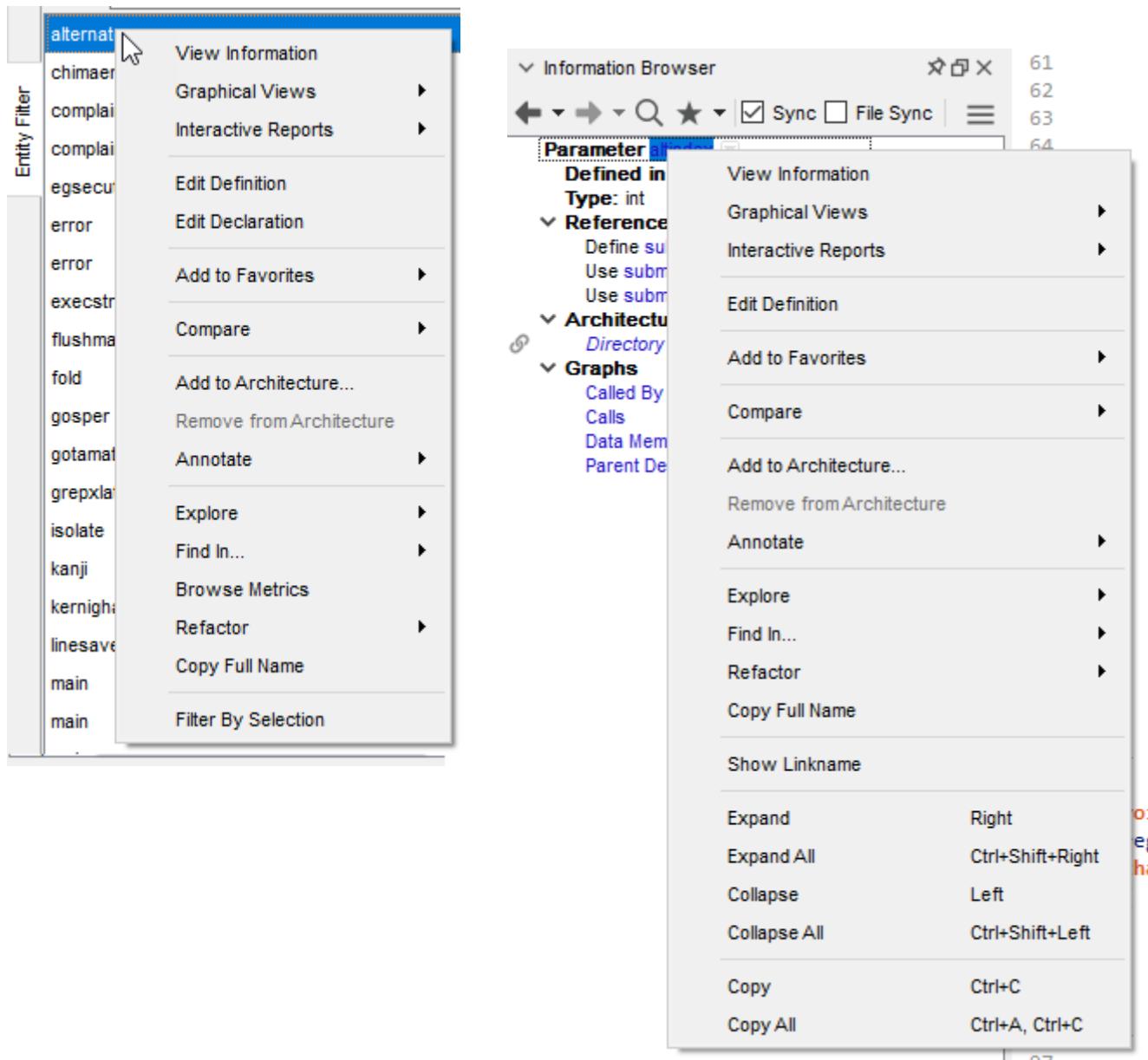
For example, if you right-click on an entity in the Source Editor, you see the list of commands shown here.

Hold down the Ctrl key while using the right-click menu to open new windows rather than re-using existing ones.

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



Right-clicking on an entity in the filter area and the Information Browser provides the following lists of options:



## Quickly Find Things in Your Source

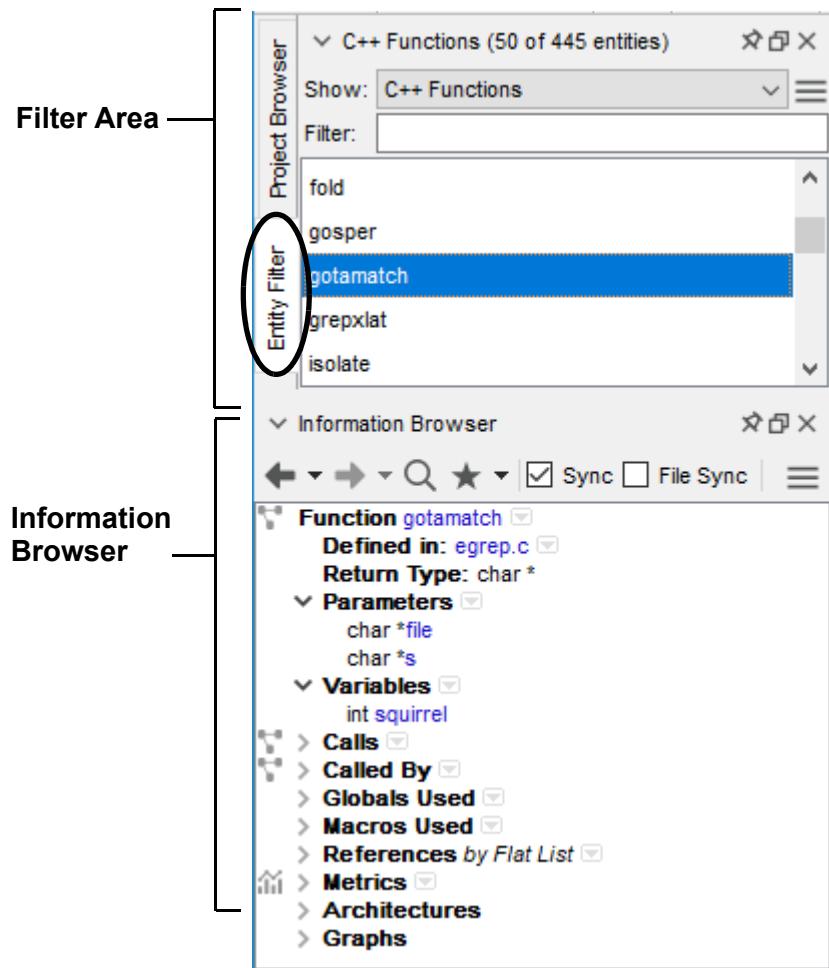
*Understand* provides several ways to quickly locate items of interest in your source code. These features include the Filter Area, the Entity Locator, and the Find in Files dialog.

### Entity Filter

The filter area of the *Understand* window helps you quickly find things in your code by separating that project into lists of Files, Code Files, Header Files, Classes, Functions, Enumerators, 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*.

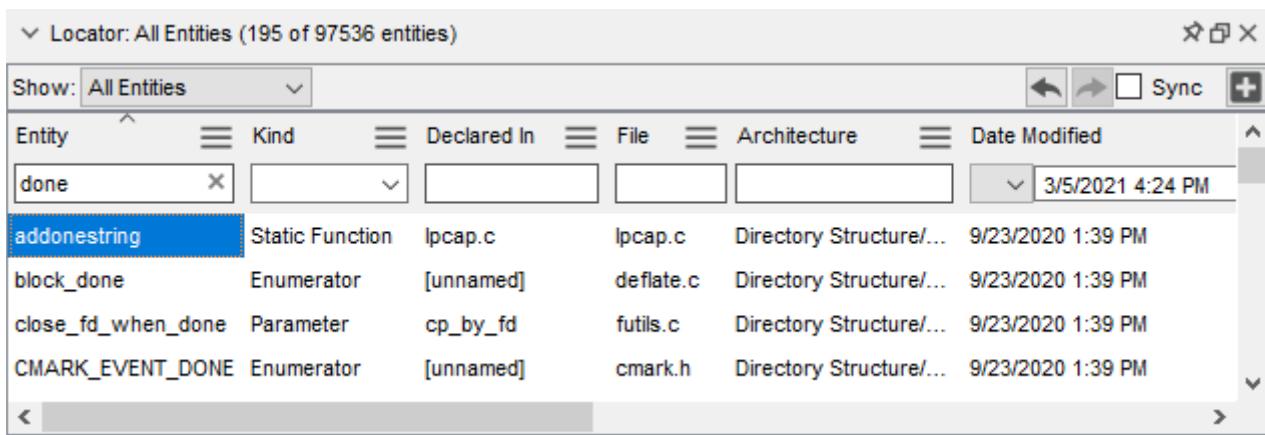


For details, see *Entity Filter* on page 124 and *Information Browser* on page 126.

**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, undefined (never declared or defined), and unresolved variables (declared but not defined) are not listed in the filters. To search or browse the entire 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 155.

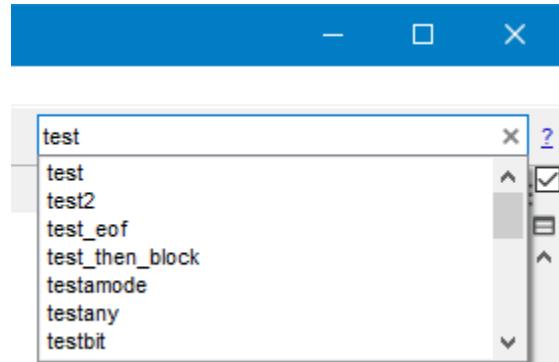
As in any other window, the context menu is also active.

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 148.



## Find in Files

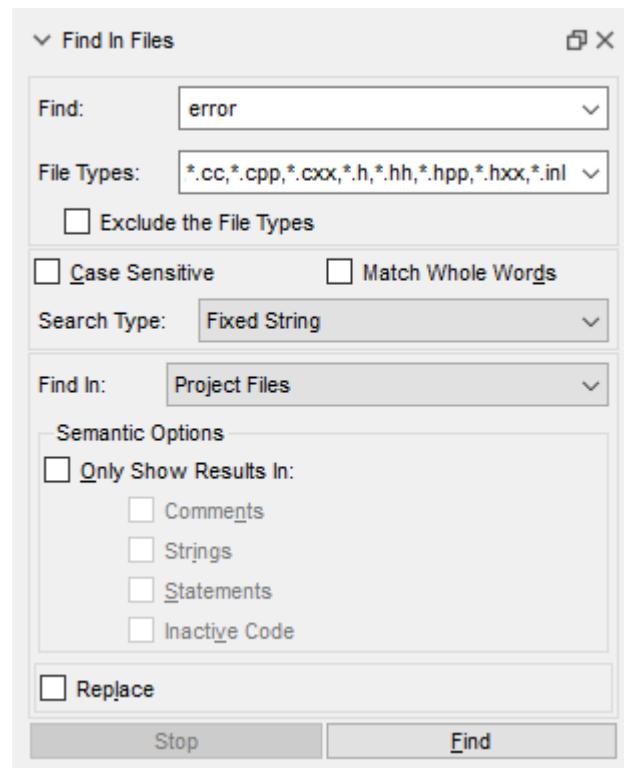
Similar to the Unix command *grep*, you may search files for the occurrence of a string.

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 *Search 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.

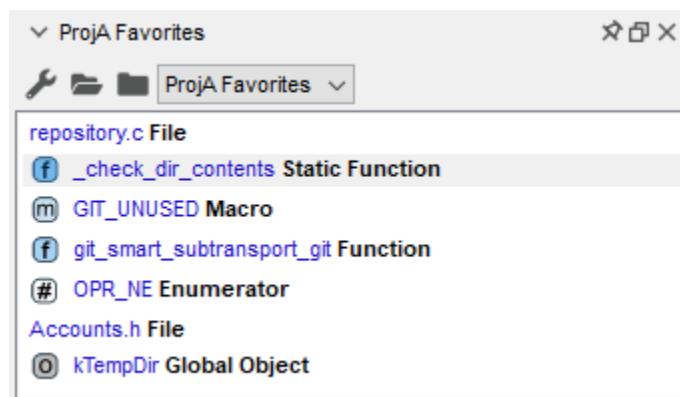
See *Find in Files* on page 150 for more information.



## 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** along with 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 142 for more information.

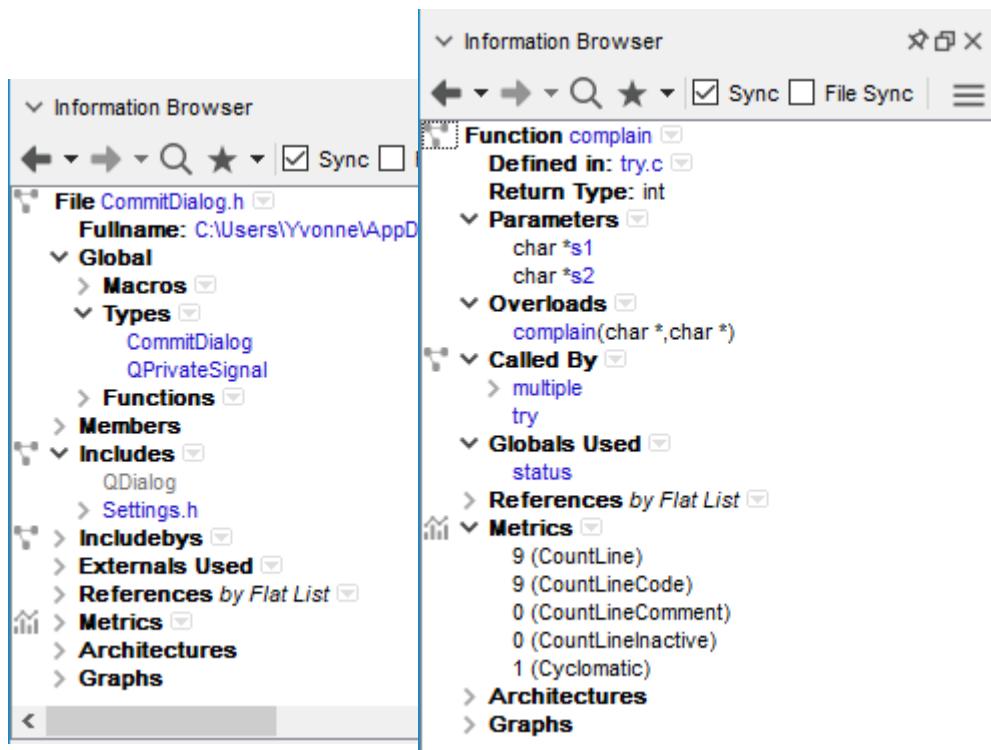
## Information Browser

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 126.

## 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.



The screenshot shows the Source Editor interface with two tabs at the top: "repository.c" and "GitAhead.cpp". The "repository.c" tab is active. The code editor displays the following C++ code:

```
9
10    #include "Application.h"
11    #include "ui/mainwindow.h"
12
13    int main(int argc, char *argv[])
14    {
15        Application::setAttribute(Qt::AA_EnableHighDpiScaling);
16        Application::setAttribute(Qt::AA_UseHighDpiPixmaps);
17        Application app(argc, argv, true);
18
19        // Restore windows before checking for updates so that
20        // the update dialog pops up on top of the other windows.
21        if (!app.restoreWindows())
22            MainWindow::open();
23
24        // Check for updates.
25        app.autoUpdate();
26
27        return app.exec();
28    }
```

The code is color-coded: comments are in green, strings are in blue, and variables and functions are in black. The line numbers are on the left. A vertical scrollbar is on the right.

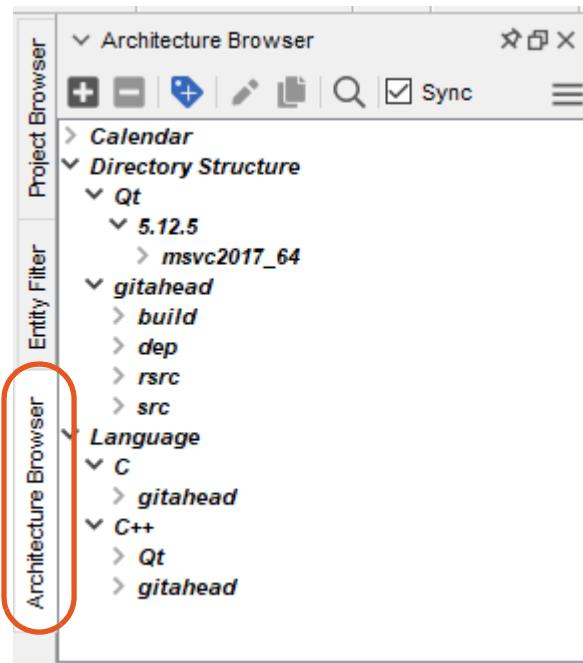
As with any other place in *Understand*, a context menu is available throughout the editor. To learn about something just right-click on it to see what information is available.

For details, see *Source Editor* on page 166.

## Architecture Browser

The Architecture Browser allows you to manage *architectures*. It shows a list of all the defined architectures in the project 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 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.

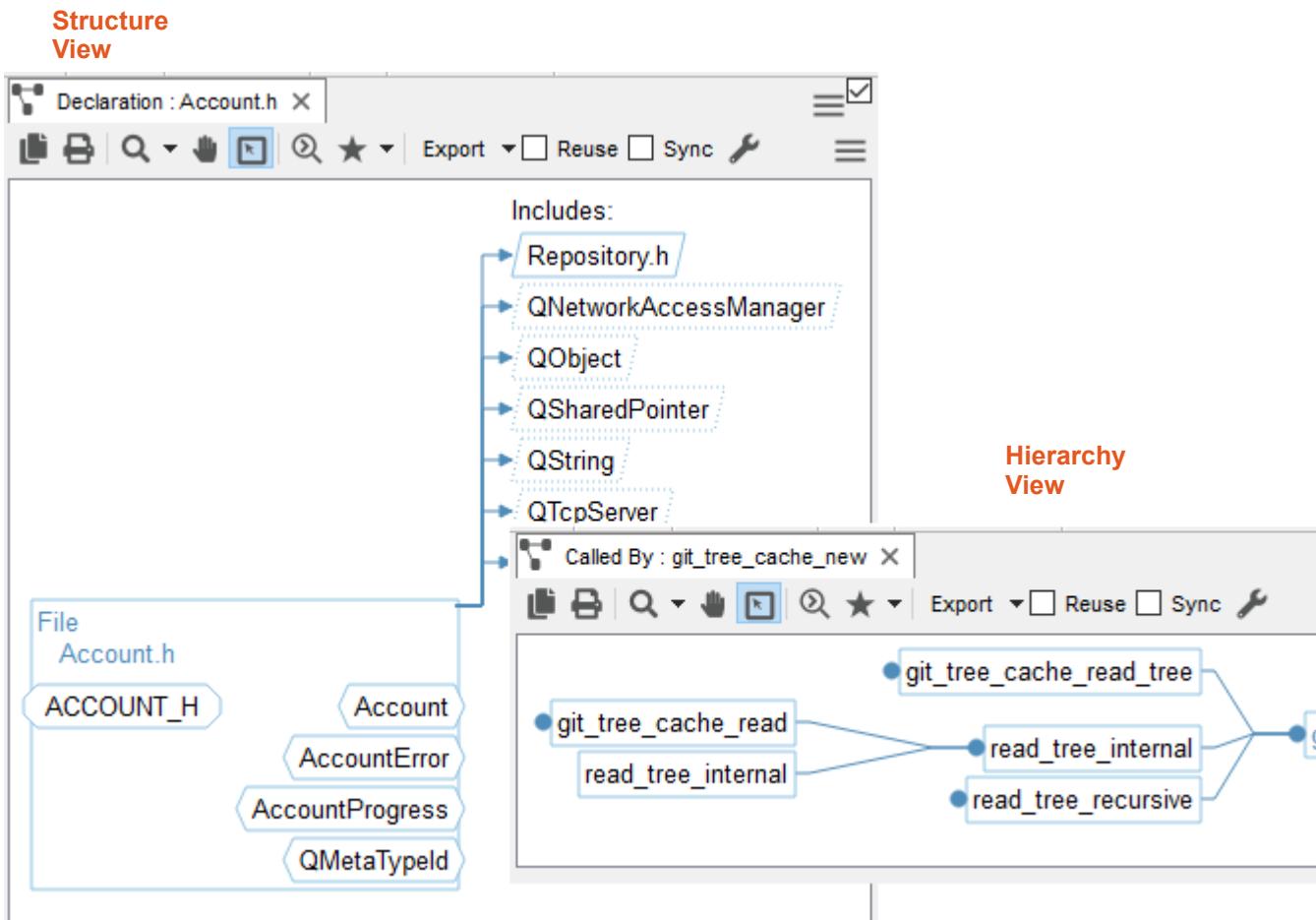
For details, see *About Architectures* on page 193.

## Graphical Views

*Understand* analyzes your software code and creates a database containing information about the entities and the relations between entities. The project 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 216.

## 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, a Java 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. Java API documentation is provided in the `doc/manuals/java` subdirectory of the *Understand* installation.

Tutorials and information about the APIs are available in the [API/Plugins category](#) on the support site.

---

## Chapter 3

# Configuring Your Project

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	32
Creating a New Project	34
Project Configuration Dialog	39
Languages Category	41
Files Category	42
File Type Options	47
File Options	48
Imports Options	49
History Options	51
Portability Options	52
Ada Options	54
Assembly Options	58
COBOL Options	60
C++ Options	62
C# Options	73
Fortran Options	74
Java Options	77
JOVIAL Options	79
Pascal Options	81
PL/M Options	83
Python Options	84
VHDL Options	85
Visual Basic (.NET) Options	86
Web Options	86
Analyzing the Code	88

## 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 analysis of the code.

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

---

### Project Storage

*Understand* projects are stored in a directory tree. All files are text-based and small so that they are easy to share and/or manage with version control systems.

Projects are auto-saved whenever they are changed. Any changes in the UI, like resizing windows, are also saved as they happen. Additionally, layouts are stored at the project level, so you can arrange windows a certain way and that project will always show up that way for you.

By default, projects you create are stored in a directory named `<project>.und` within the top level source code directory for the project. The files in this directory contain all the information needed (aside from the source code) to recreate the project.

If you want to make a copy of the current configuration, for example to create two variants of one configuration, you can make a copy of the directory tree where your project settings are stored.

Files that are not usually shared or managed as source code are stored in a separate location. These files include user-specific data, such as favorites and bookmarks, and the `parse.udb` database, which is stored in a proprietary binary format. You can find the location where such internal project information is saved by running the following command line: `und projectinfo myproject.und`. The default locations are:

- **Windows:** `C:\Users\<Username>\AppData\Local\SciTools\Db`
- **Linux/Unix:** `~/.local/share/Scitools/Db`
- **MacOS:** `~/Library/Application Support/Scitools/Db`

If desired, you can force *Understand* to store all files related to the project in the `<project>.und` directory. Do this by creating an empty directory named “local” inside the `<project>.und` directory, which will be populated with other files.

The `parse.udb` project database permits multiple simultaneous read accesses, but it does not support multi-user write access. You will see a message if the project database is locked.

Occasionally, a new feature to *Understand* requires a change to the database format. Such changes are noted in the Build Notes for that version. When you install a build that modifies the database format, existing projects are automatically re-analyzed when you open them.

Sample projects that you download and open from the Welcome page are typically stored in the following location:

- **Windows:** `C:\Users\<Username>\AppData\Roaming\SciTools`
- **Linux/Unix:** `~/.config/SciTools`
- **MacOS:** `~/Library/Application Support/SciTools/samples`

## Creating a New Project

To begin analyzing code, you create a project and specify what source files to analyze. *Understand* analyzes your code and stores the information. This information can be refreshed incrementally in the GUI or updated using command-line tools.

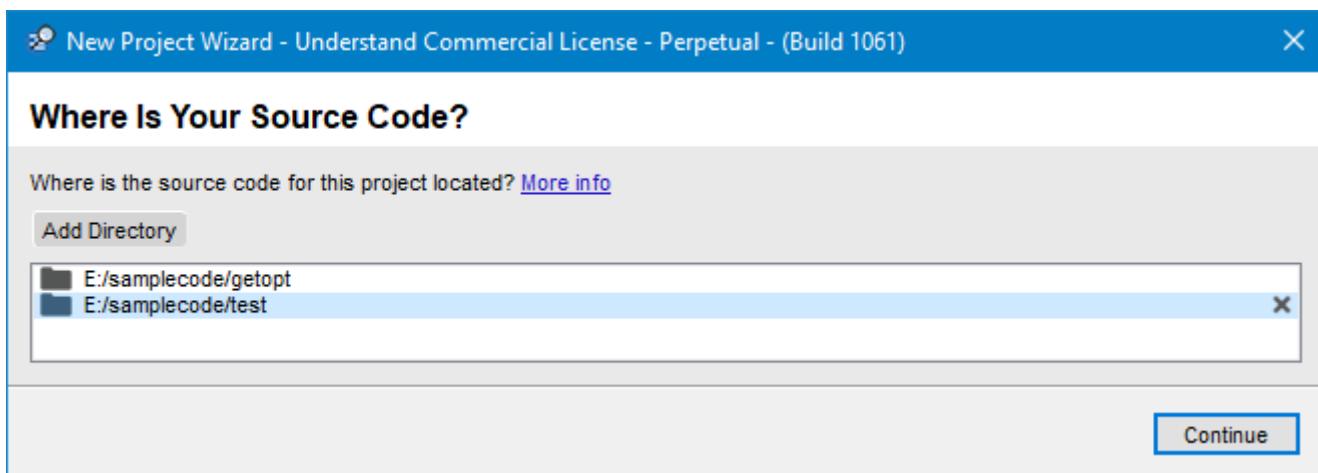
To create a new project, follow these steps:

- 1 Click the **New** link in the Welcome page you see when you start *Understand*. Or, choose **File > New > Project** from the menus.
  - By default, this opens the New Project Wizard.
  - Alternately, you may have disabled the option to run this wizard (see page 96), in which case, you see the “Create new project as...” dialog. Browse to the folder where you wish to create the project. Type the name of the project in the **File name** field. Click **Save**. A directory with the name you provided and a suffix of “.und” will be created in the selected location. 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 296.

**Note:** Any project that was previously open is closed when you start to create a new project. You can only open one project at a time per instance of *Understand*.

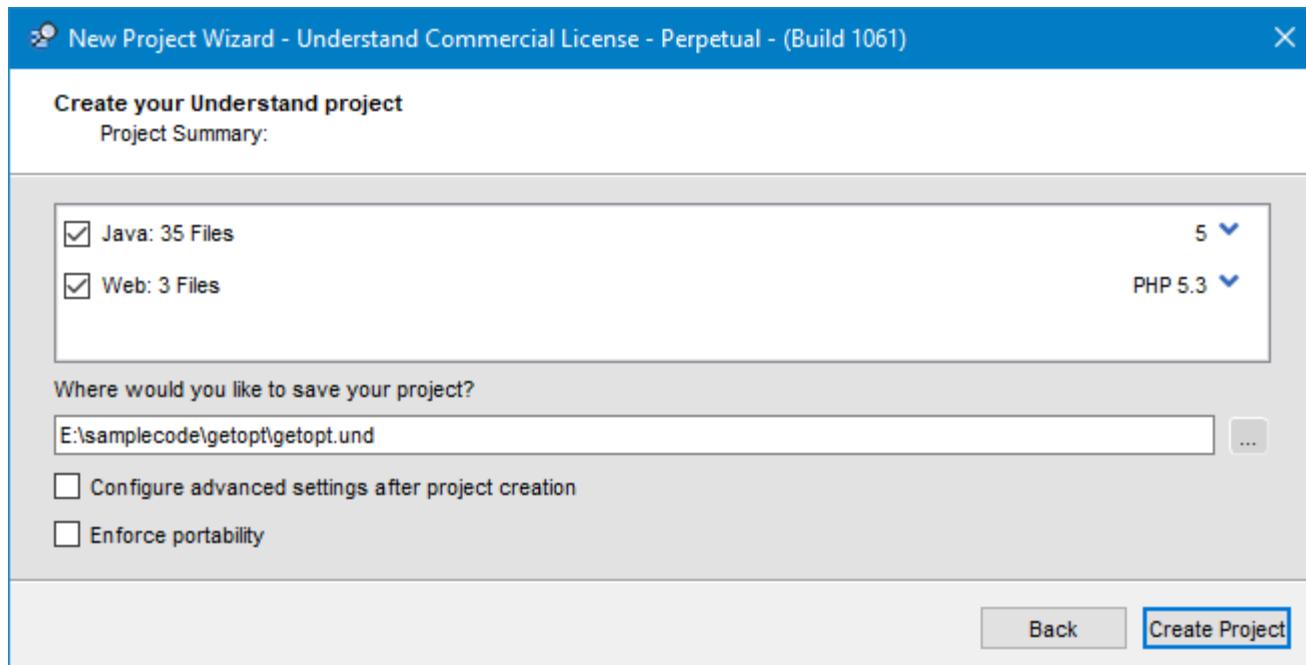
- 2 In the first page of the wizard, click **Add Directory** and browse to the top-level directory of your source code. You can select one or more root directories.

**Tip:** If your project contains multiple executable files, it is recommended that you create a separate *Understand* project for each executable.



- 3 Click **Continue**.

- 4 Other prompts may occur at this point depending on the contents found in your source directory. Answer these prompts as appropriate and click **Continue**.
  - For example, if an *Understand* project already exists within this location, you see prompts asking whether you want to use the settings in that project.
  - If build files exist, you are asked if you want to allow *Understand* to watch your build to create a more accurate project. See *Using Buildspys to Create Projects* on page 36.
  - If the source files use Cmake, you are asked if you want to import information from the Cmake compilation database file. See *Creating Projects from Cmake Projects* on page 36.
  - If a Visual Studio project exists within the location, you are asked if you want to import it to create a more accurate configuration. See *Creating Projects from Visual Studio Projects* on page 37.
  - If an Apple Xcode project exists within the location, you are asked if you want to import it to create a more accurate configuration. See *Creating Projects from Apple Xcode Projects* on page 38.
- 5 The “Create Your *Understand* Project” page of the wizard detects the languages used in your source code and provides a default location for the *Understand* project files.



- 6 You can disable languages using the checkboxes on the left. (If you want to re-enable a language later, you can do that in the Project Configuration settings.)
- 7 For most languages, you can select the version of the standard or the compiler you want to use as the basis for analyzing the code from the drop-down list on the right.

- 8 By default, the project will be placed in a directory within the main source code directory. *Understand* project files are text-based and small so that they are easy to share and/or manage with version control systems. If you do not want to store the project with your source code, click ... next to **Where would you like to save your project** and browse to select another location. See page 32 for more about how projects are stored.
- 9 If you know you want to configure additional project settings—such as languages used, file types, file encoding, or language-specific options—check the **Configure advanced settings after project creation** box to automatically open the Project Configuration dialog after creating the project. See page 39 for details.
- 10 If you want file paths in the project to be stored so that you can share the project with other users, check the **Enforce portability** box. See *Portability Options* on page 52 for details.
- 11 Click **Create Project**. The new project is created and opened.

---

## Using Buildspy to Create Projects

When you import a project that contains C/C++ code the New Project Wizard asks if you want to create a more accurate project by allowing *Understand* to watch your build process. It can do this by analyzing information about include paths and macro definitions that are passed to the compiler.

To use this feature, choose **Yes, watch me build my C/C++ project** and follow the prompts.

You should first clean your build or delete the build directory when prompted so that your entire compilation process will run. When you run the build, build against a single target at a time instead of compiling for multiple targets at once. You can run multiple builds while Buildspy is collecting compiler actions.

Buildspy detects compilers with a command of `c++`, `cc`, `cl`, `clang`, `clang++`, `g++`, and `gcc`. If your compiler has a different command, click **Watch Options** before starting the build and add your compiler command.

To run Buildspy from the command line, see *Using Buildspy to Build Understand Projects* on page 296.

---

## Creating Projects from Cmake Projects

If you use Cmake to build your projects, you can use it to generate your *Understand* project. It will add all of the files, set up the correct macro definitions for each file, and set up the correct include files for each file in the project. Projects created this way will be much more accurate than projects created by hand, allowing you to have easier access to all the *Understand* features.

Follow these steps to use Cmake files to build a project:

- 1 Navigate to your build directory.
- 2 Edit the `CMakeCache.txt` file (in a build folder) and set `CMAKE_EXPORT_COMPILE_COMMANDS` to ON.
- 3 Clean the build so that `make` will run a full build.
- 4 Run `make` for the build target whose `CMakeCache.txt` file you edited. A file called `compile_commands.json` is generated in your build directory.

- 5 In *Understand* choose **File > New > Project** to run the New Project Wizard.
- 6 Specify the root directory of a project that uses Cmake and contains a `CMakeLists.txt` file and has one or more builds set up. See page 34 for details.
- 7 When you click **Continue**, *Understand* detects that the source uses Cmake and asks if you want to specify a Cmake compilation database.
- 8 Click **Add File** and choose the `CMakeCache.txt` file that was generated.

## Specify CMake Compilation Database?

We notice this project uses CMake but we did not find a compilation database file. This file (`compile_commands.json`) will allow for a ***much more accurate*** *Understand* project. Would you like to specify a compilation database file (instructions below)?

**Add File**

How To Create a CMake Compilation Database:

1. First, find and open the file `CMakeCache.txt` in the root of your build folder (i.e. `C:/Code/build/release/CMakeCache.txt`).
2. Change the value for `CMAKE_EXPORT_COMPILE_COMMANDS` to 'On' and save and close the file.
3. Run the clean build target (e.g. 'make clean' or 'ninja clean')
4. Run the build target for the desired project (e.g. 'make myProject' or 'ninja myProject')
5. The file `compile_commands.json` should now show in the root of your build folder (i.e. `C:/code/build/release/compile_commands.json`)

**Back**

**Skip**

- 9 Click **Continue**. You will see the "Create Your Understand Project" page of the wizard, which detects the languages used in your source code and provides a default location for the *Understand* project files. See page 35 for the remainder of the project creation process.

For information about using the command line to create *Understand* projects from source projects that use Cmake, see the "Cmake and Understand" topic on the [Support website](#).

### Creating Projects from Visual Studio Projects

If you use Visual Studio to build your projects, you can use its project files to generate your *Understand* project. It will add all of the files, set up the correct macro definitions for each file, and set up the correct include files for each file in the project. Projects created this way will be much more accurate than projects created by hand, allowing you to have easier access to all the *Understand* features.

Follow these steps to use Visual Studio files to build a project:

- 1 In *Understand* choose **File > New > Project** to run the New Project Wizard.
- 2 Specify the root directory of a project that uses Visual Studio. See page 34 for details.

- 3 When you click **Continue**, *Understand* detects that the source uses Visual Studio and asks if you want to import files from other tools to improve project accuracy.

The files suggested may include .csproj files, which contain the list of files included in the project along with references to system assemblies. These files are created by Visual Studio when a new project or assembly is created. If you have multiple assemblies, you will have multiple .csproj files. The solution file ties the various .csproj files that make up the project together.

- 4 Click **Import** next to the solution file (.sln).
- 5 Use the drop-down **Configuration** list to specify the configuration that the *Understand* project should match.
- 6 Click **Continue**. You will see the “Create Your Understand Project” page of the wizard, which detects the languages used in your source code and provides a default location for the *Understand* project files. See page 35 for the remainder of the project creation process.

For more information about using *Understand* with Visual Studio, see the [Support website](#).

---

### Creating Projects from Apple Xcode Projects

*Understand* detects whether the source code for a project you are creating is managed by the Apple Xcode IDE. Follow these steps when creating an *Understand* project from an Xcode project:

- 1 In *Understand* choose **File > New > Project** to run the New Project Wizard.
- 2 Specify the root directory of a project that uses Xcode. See page 34 for details.
- 3 When you click **Continue**, *Understand* detects that the source uses Xcode and asks if you want to import files from other tools to improve project accuracy.
- 4 Click **Import** next to the suggested Xcode project file, which may have a .xcodeproj file extension. Nested Xcode projects are supported.
- 5 Use the drop-down **Configuration** list to specify the configuration that the *Understand* project should match.
- 6 Click **Continue**. You will see the “Create Your Understand Project” page of the wizard, which detects the languages used in your source code and provides a default location for the *Understand* project files. See page 35 for the remainder of the project creation process.

For more information about using the *Understand* with Xcode, see the [Support website](#).

## Project Configuration Dialog

The Understand Project Configuration dialog opens when you choose the **Project > Configure Project** menu item.

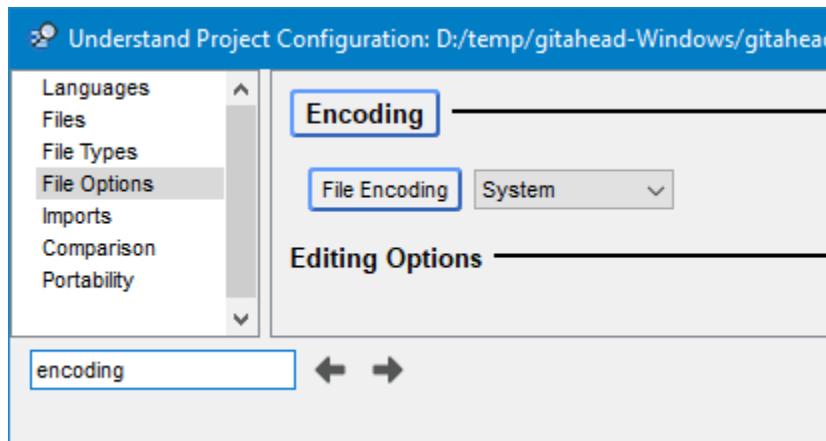
The categories on the left in the Project Configuration dialog allow you to specify various project settings to be used during analysis. The Project Configuration dialog contains the following categories:

- **Languages:** Set the types of languages to be analyzed. See [page 41](#).
- **Files:** Set the locations of source files to be analyzed. See [page 42](#).
- **File Types:** Set how to handle source file types and what file extensions are used. See [page 47](#).
- **File Options:** Set the file encoding and editing mode for source files. See [page 48](#).
- **Imports:** Specify import file and exclude filters. See [page 49](#).
- **History:** Select a Git repository and/or previous project for history information and entity comparison. See [page 51](#).
- **Portability:** Select project portability options. See [page 52](#).
- **Language-Specific Options:** Set options for the languages you selected in the Languages category. For details, see:
  - Ada Options, [page 54](#)
  - Assembly Options, [page 58](#)
  - COBOL Options, [page 60](#)
  - C++ Fuzzy Options, [page 62](#)
  - C++ Strict Options, [page 70](#)
  - C# Options, [page 73](#)
  - Fortran Options, [page 74](#)
  - Java Options, [page 77](#)
  - JOVIAL Options, [page 79](#)
  - Pascal Options, [page 81](#)
  - PL/M Options, [page 83](#)
  - Python Options, [page 84](#)
  - Visual Basic Options, [page 86](#)
  - Web Options, [page 86](#)

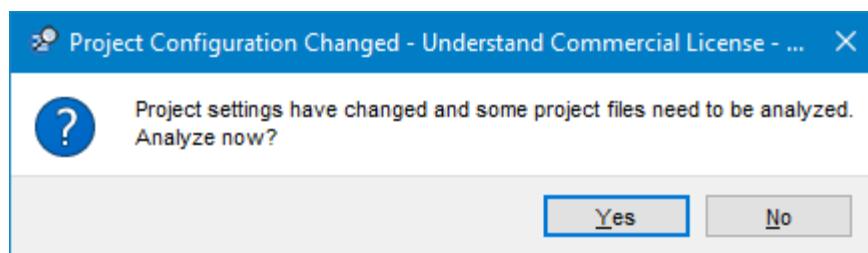
 Understand
Languages
Files
File Types
File Options
Imports
History
Portability
Ada
Assembly
COBOL
C++
C#
Fortran
Java
JOVIAL
Pascal
PL/M
Python
Visual Basic
Web

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

Since there are many configuration options, *Understand* makes it easy to find options using the Search Settings box. For example, to find the setting for File Encoding, type “encoding” in the search field. *Understand* moves to the first category that contains the search string and highlights the field. You can click the arrows to move to the next category as needed. For example:



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.



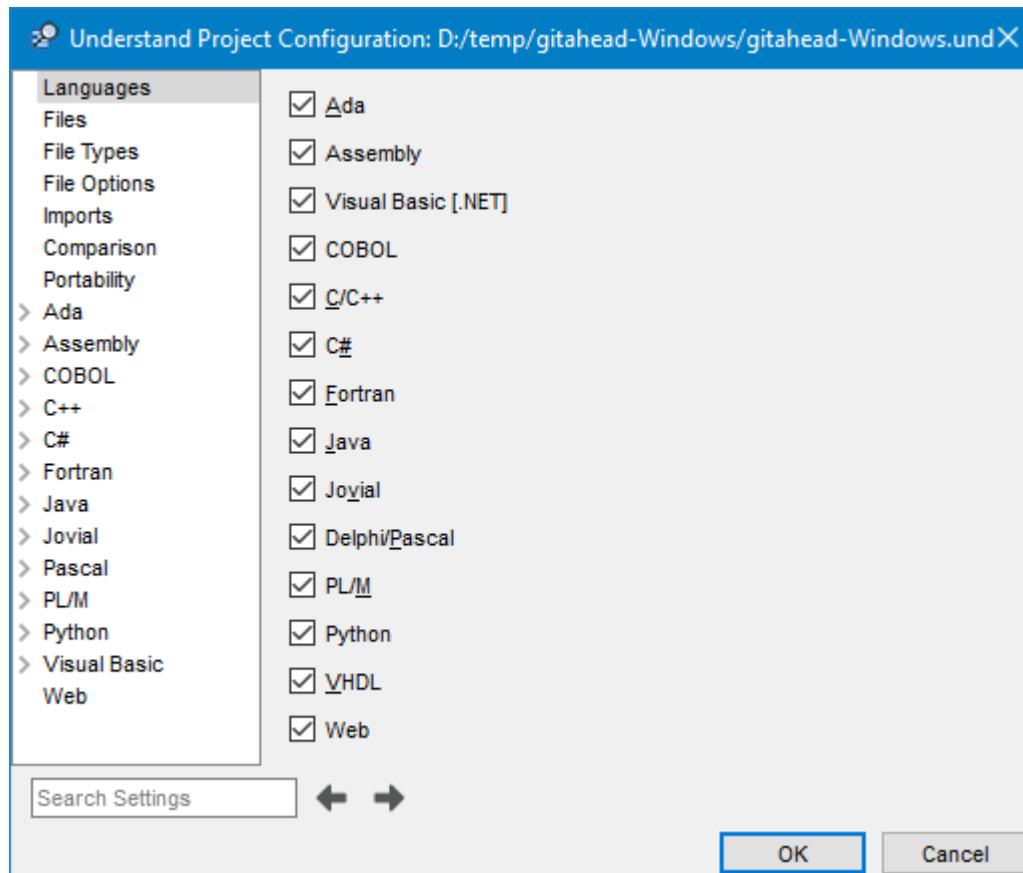
Click **Yes** and *Understand* begins analyzing (parsing) the code (page 88).

If you want to close the Project Configuration dialog without saving any changes, click **Cancel**, and then click **Discard Changes** in the box that asks if you really do not want to save your changes.

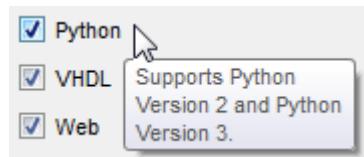
If you want to make a copy of the current configuration, for example to create two variants of one configuration, you can make a copy of the directory tree where your project settings are stored (page 32).

## 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.



For information about language support for a particular language, hover your mouse cursor over the language name.

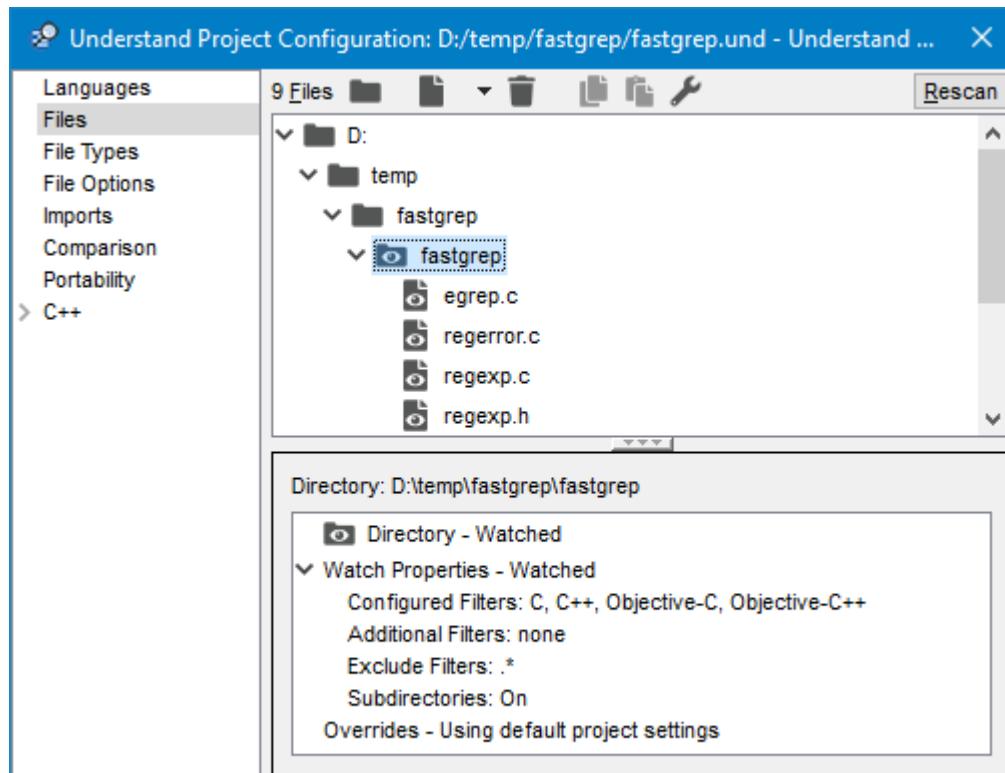


When you select a language, categories for that language are added to the list on the left in the Project Configuration dialog. The languages you choose here not only affect how the source files are analyzed. They also affect the filter types available and the metrics available.

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.

## 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 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). You can also synchronize a project with a CMake or Xcode project. See *Creating Projects from Cmake Projects* on page 36, *Creating Projects from Visual Studio Projects* on page 37, and *Creating Projects from Apple Xcode Projects* on page 38.

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.

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

- Open the Add a Directory dialog.
- Open the Add Files dialog.
- Choose to 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.

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

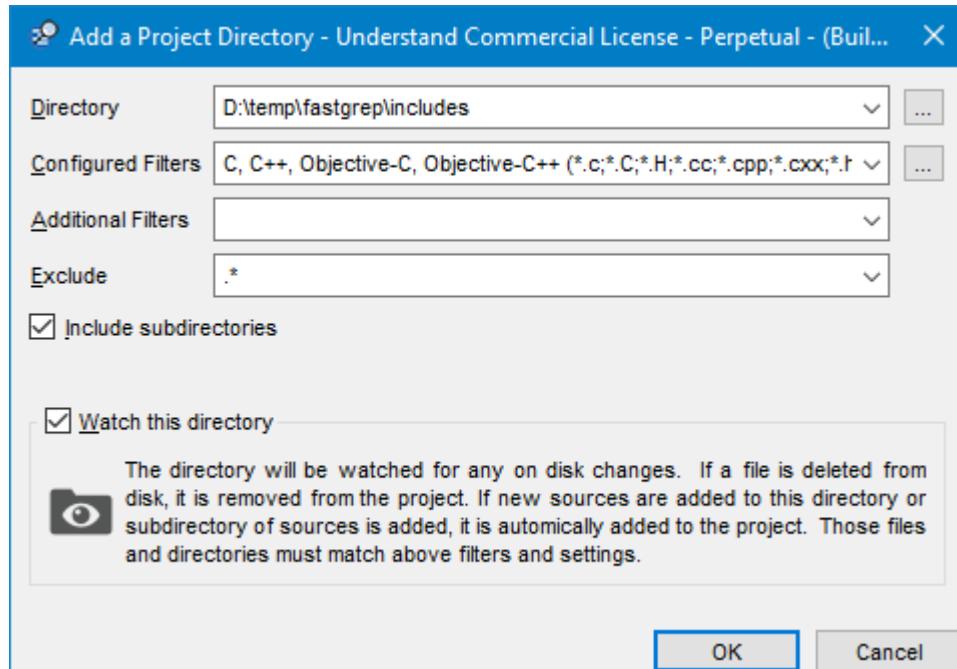
Click **Rescan** if you have added files to a directory that are not shown in this dialog.

Right-click for options to add or remove a directory or files, expand or collapse the directory tree, rescan the directory for changes, and configure the override settings.

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

## 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 Basic, MSDos Batch, Perl, SQL, Tcl, Text, Verilog, and XML.  
If this directory contains source files with extensions that are not listed, click **Configure**. Also, see *File Type Options* on page 47. 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 an eye  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.

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.

Directory paths are displayed as absolute paths in the interface but in most cases are stored as relative paths in the project.

---

## 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 , 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 289 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 directory or file itself is not deleted from the file system. The filenames of removed files are shown with a strike-through.

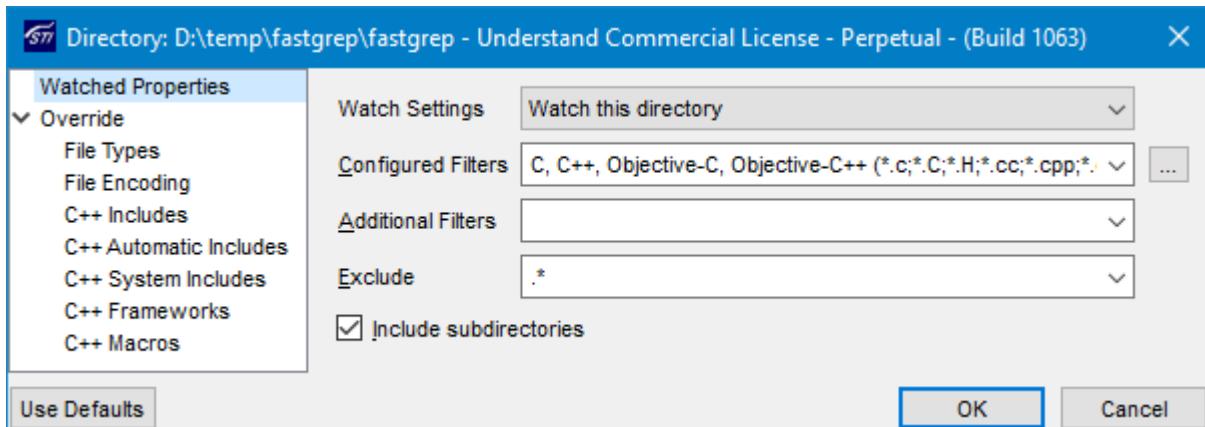
You can right-click on a removed file or directory and choose **Restore Selected Files** to re-add it to the project.

## 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.

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

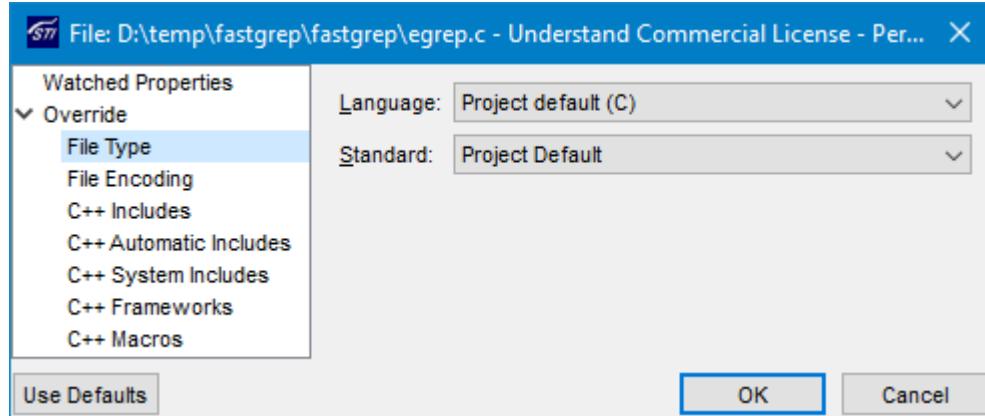
- 1 Select a directory.
- 2 Click  or right-click and select **Configure override settings**.



- 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 **Watch this directory**, **Do Not Watch directory**, or **Use Parent directory settings**. 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 as described in *Adding Directories* on page 43.
- 4 The **File Type** category lets you override the languages used for files in this directory as described in *File Type Options* on page 47. The **File Encoding** category lets you override the encoding setting described in *File Options* on page 48.
- 5 In the various language-specific override categories, you can make directory-specific language-related settings. The categories available are different depending on the language of the source file. See page 54 through page 83 for details.
- 6 Click **OK** to save your overrides.

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

- 1 Select a file.
- 2 Click  or right-click and select **Configure Override Settings**.



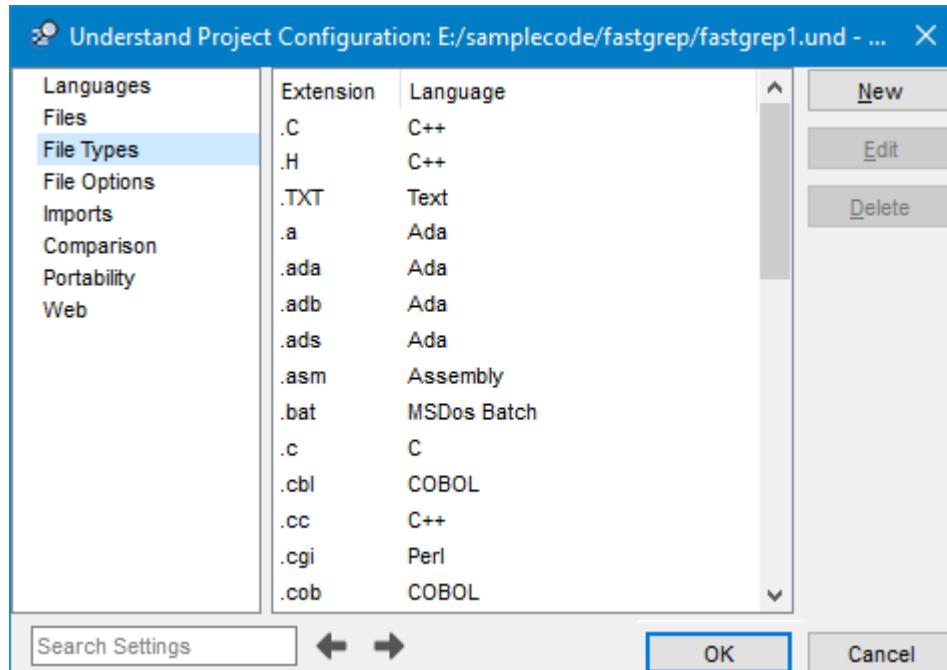
- 3 In the **Watched Properties** category, you can choose **Watched by parent directory** to have the file watched if the parent directory is watched or Exclude from watch.
- 4 The **File Type** category lets you override the languages used for the file as described in *File Type Options* on page 47. The **File Encoding** category lets you override the encoding setting described in *File Options* on page 48.
- 5 In the various language-specific override categories, you can make file-specific language-related settings. The categories available are different depending on the language of the source file. See page 54 through page 83 for details.
- 6 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.

## File Type Options

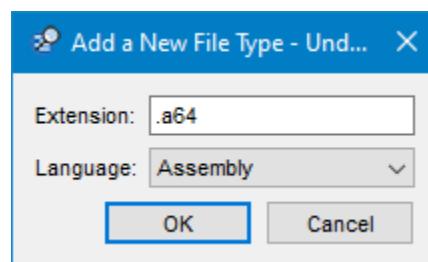
In the **File Types** category of the Project Configuration dialog, you can control how file extensions are interpreted by *Understand*.



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**.



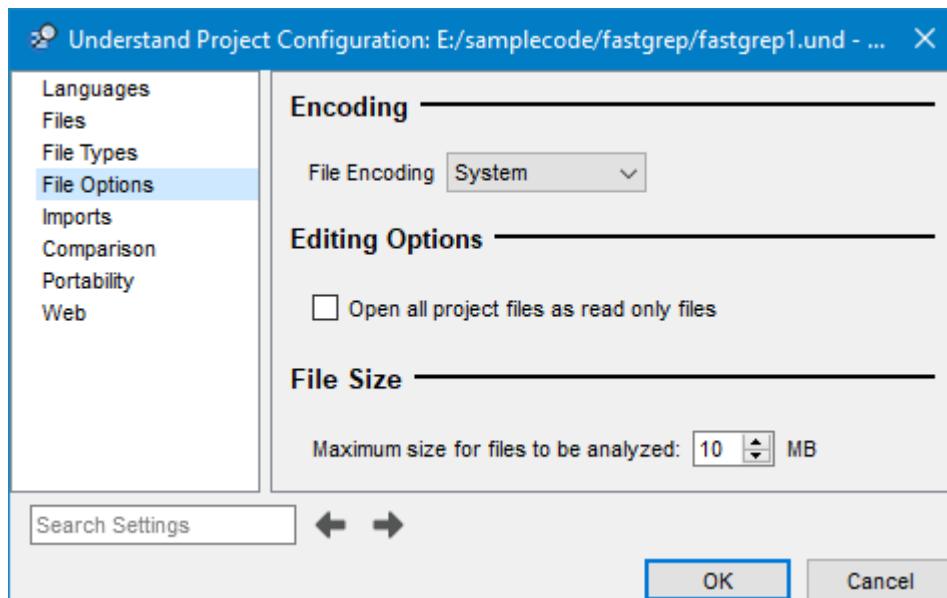
The file extension should begin with the period. It can contain simple \* and ? wildcards.

Certain file types may be interpreted differently depending on the languages you selected. For example, in a Visual Fortran project, .h files are interpreted as Fortran files, rather than as C headers files.

You can override the file type settings on a file-by-file or directory-by-directory basis (see *Setting Overrides* on page 45).

## File Options

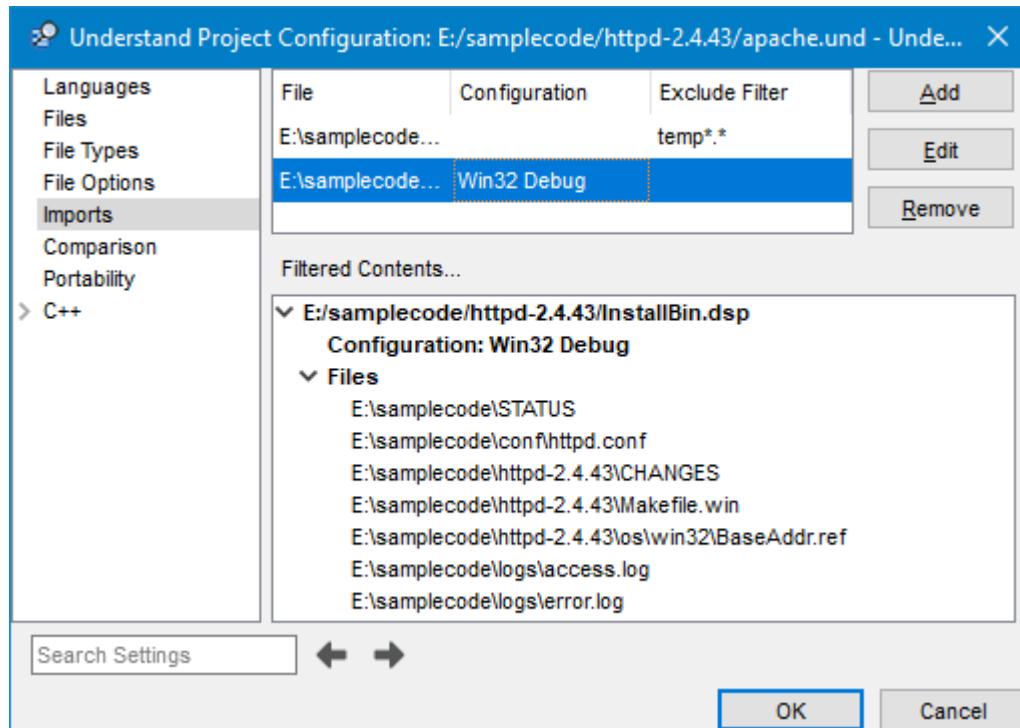
In the **File Options** category of the Project Configuration dialog, you can control how files are opened and saved by *Understand*.



- **File Encoding:** Select the type of encoding to use when saving source files for this project. Many encoding formats are supported. You should change this setting at the project level if you want it to be different than the setting for other projects or 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. 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 in this project to be edited and saved within *Understand*.
- **Maximum size for files to be analyzed:** Limits the size of files analyzed by *Understand*. You can use this option to exclude very large files. The default is 10 MB. An error message is provided if you attempt to edit a file that is too large to open.

## Imports Options

You can import various file types from your source projects to configure the list of directories and files in your *Understand* project.



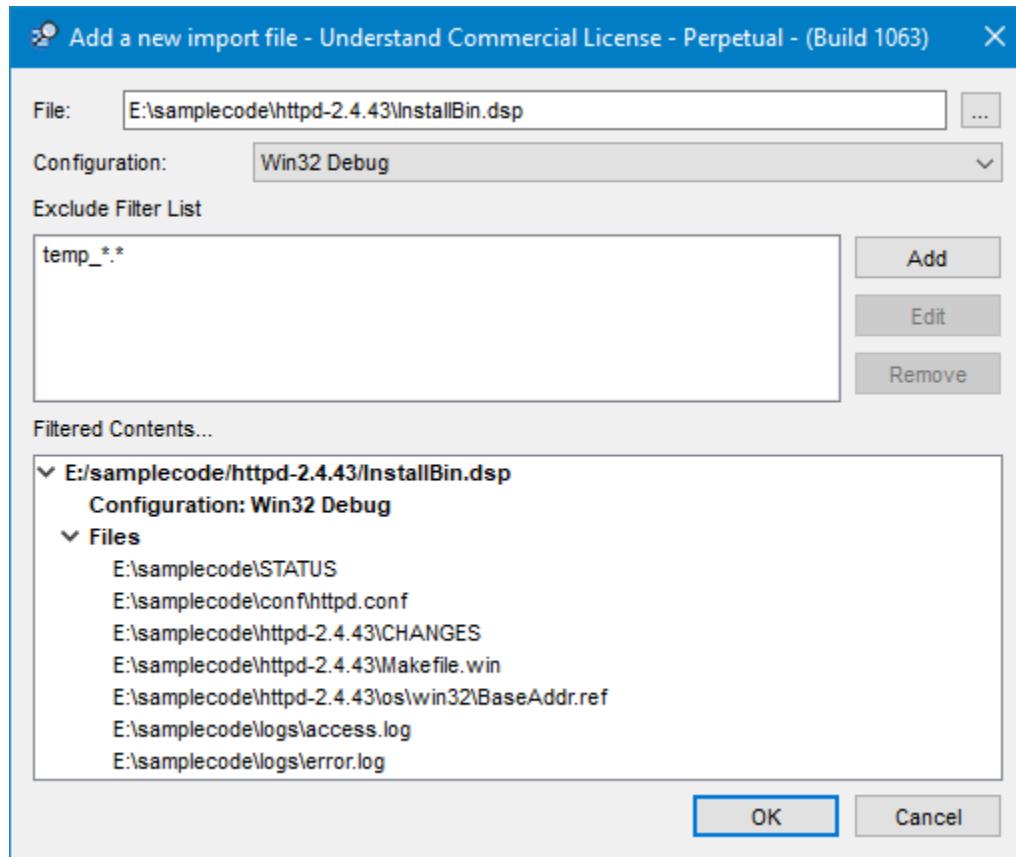
The following file types can be imported:

- **JSON compile command file:** `compile_commands.json`. See *Creating Projects from Cmake Projects* on page 36.
- **Visual Studio project files** (including some older formats): `*.csproj`, `*.dsp`, `*.dsw`, `*.sln`, `*.vbproj`, `*.vcp`, `*.vcproj`, `*.vfproj`, `*.vcw`, `*.vcxproj`. See *Creating Projects from Visual Studio Projects* on page 37.
- **Apple Xcode project files:** `*.xcodeproj`. See *Creating Projects from Apple Xcode Projects* on page 38.

To import a file, follow these steps:

- 1 In the **Imports** category of the Project Configuration dialog, click **Add**.
- 2 Click ... to browse for a file of one of the types listed above.

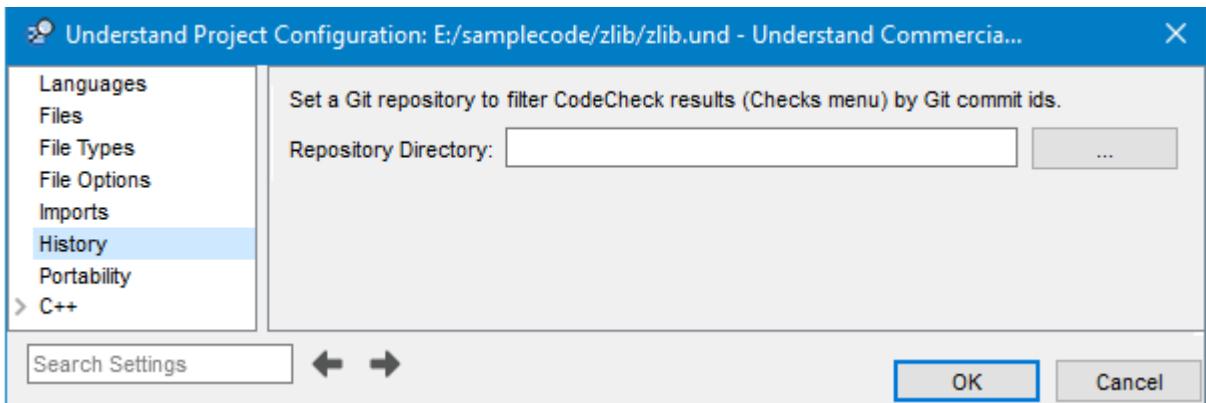
- 3 Select the file and click **Open**. You should see a list of the project settings in the file you selected.



- 4 If you want to exclude any files in *Understand* that are part of your source project, click **Add** and type a string that uses wildcard matching.  
5 Click **OK** to save your changes.  
6 You are asked if the project should be analyzed. Click **Yes** to update the project.

## History Options

A project can get history information from a Git repository and/or by comparing a previous version of the *Understand* project.



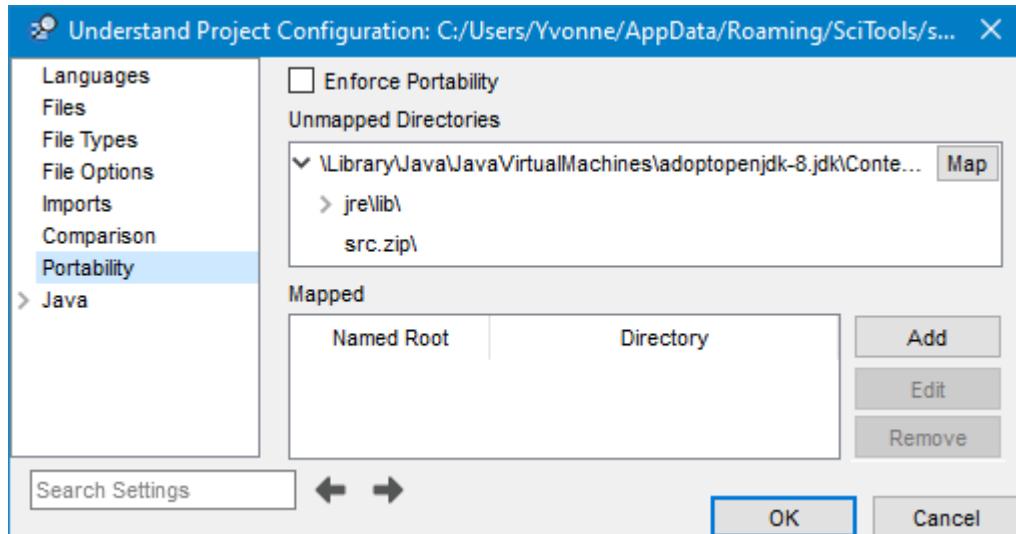
- **Repository Directory:** If the files in your project are managed using a Git repository, most Git-related features, such as displaying “blame” information in the Source Editor windows (page 267), will work automatically. However, to use Git filtering with CodeCheck (page 245), you should point to the folder that was cloned from the Git repository.

Click the ... button and select the top-level cloned folder in your Git repository.

To point to a previous version of your project for comparison graphs, see page 266.

## Portability Options

You can control the portability of *Understand* projects using the **Portability** category in the Project Configuration dialog. A portable project allows you to share the project with other users and to use the project unchanged after moving the source code files.



Projects use relative paths by default to make them more shareable. However, if the .und project folder and source code are in different root folders or on different drives, absolute paths are used. To make such projects shareable, create Named Roots.

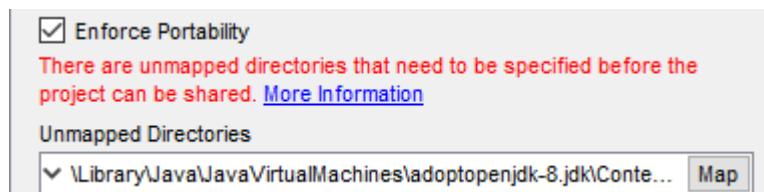
For example, C:\src\proj1 and C:\projects\proj1.und are in different root folders, so the project would use absolute paths. In contrast, C:\src\proj1 and C:\src\projects\proj1.und are in the same root folder (C:\src), so the project would use relative paths.

If you create projects in the default location, which is the root of the source tree, projects should always be stored with relative paths and shareable automatically.

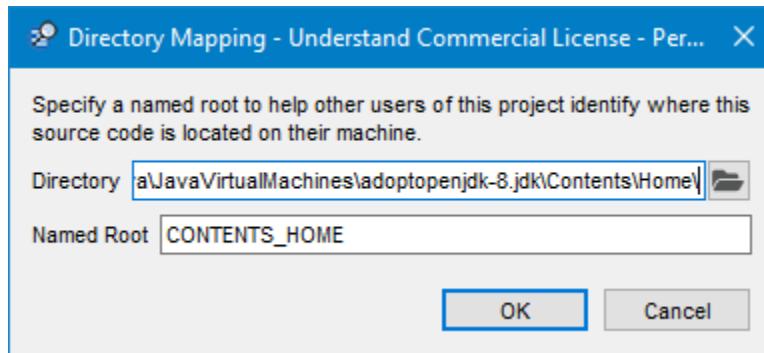
If you check the **Enforce Portability** box, the *Understand* project files will contain relative paths to all directories. These paths are relative to the location of the *Understand* project (*project.und* directory). If you store the project in the source file tree and move it along with the source files, the project can still be used.

In order to make a project portable, any directories that are not part of the source tree but are referenced in the project should be mapped using named roots. For example:

- 1 In the **Portability** category of the Project Configuration dialog, check the **Enforce Portability** box.
- 2 If there are any unmapped directories, you will see a message like this:



- 1 Click the **Map** button next to a top-level unmapped directory. A suggested named root will be provided. This example shows the suggested named root for that Java Virtual Machines directory.



- 2 Change the **Named Root** as desired (for example, if you have an environment variable customarily used to point to this directory).
- 3 You can change the **Directory** to point to the location of this directory on your system.
- 4 Click **OK**.
- 5 When there are no remaining unmapped directories listed in the Project Configuration dialog, click **OK**.
- 6 When you share the project, tell other users to use this dialog to point to their own locations of these directories.

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 287). This is useful, for example, if you want to share projects with people who reference project files over a network using different paths.

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 variables 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:

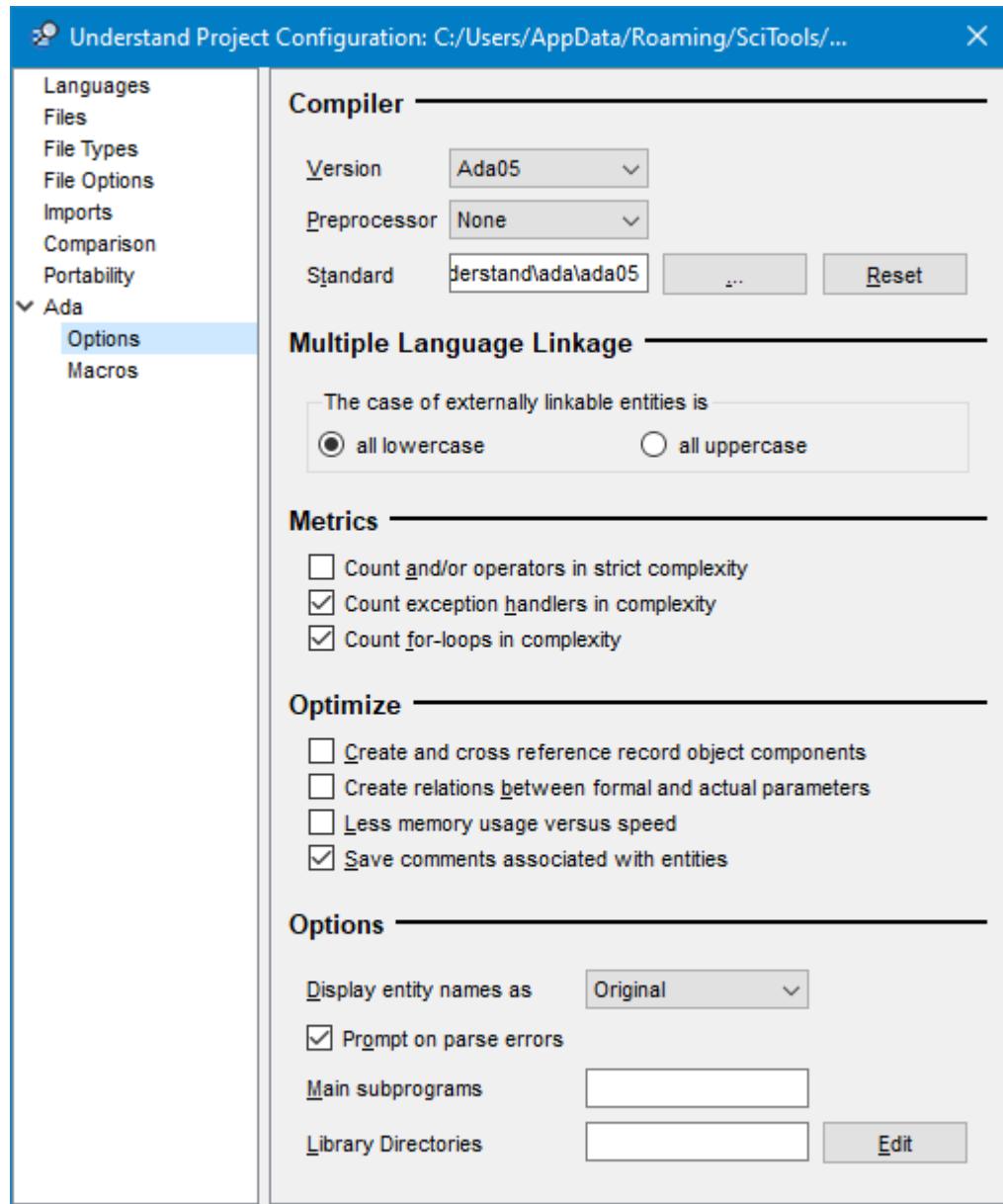
```
UND_NAMED_ROOT_SOURCEDIR=c:\my\project\dir
```

The named root that you would then use within *Understand* is “SOURCEDIR”.

If you are using the “und” command-line tool, named roots definitions on the “und” command line have the highest precedence. The next precedence is named roots defined as environment variables at the operating system level, and finally by named roots defined in the *Understand* project configuration.

## 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.



The fields in this category are as follows:

- **Version**: Choose the version of Ada used in your project. *Understand* supports Ada83, Ada95, Ada05, and Ada12.

- **Preprocessor:** Choose which type of preprocessor statements are used in your Ada code. The choices are None, C, Gnatprep, Green Hills, and Verdix. Note that if your source code directories contain a Gnat \*.gpr project file, that file will be analyzed whether or not you select the Gnatprep preprocessor.
- **Standard:** You may choose a directory that contains a standard Ada library used by this project.

Sometimes it is helpful to analyze code in the 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.

The custom standard directory must include a “standard” package. If your compiler’s specification files do not include a standard package file, copy a standard file from one of the directories provided with *Understand*. For example, for an Ada05 project, copy the `<install_dir>/conf/understand/ada/ada05/standard05.ads` file.

- **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:** Check box if source code comments before and after an entity should be associated with that entity. On by default.
- **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 analyzed as part of a project. All subdirectories of the directory you select will also be used to find libraries.

## Ada > Macros Category

This category allows you to define macros to match the macros defined when your project is compiled. Projects may use such macros with the selected preprocessor (page 54) and/or with Ada pragmas.

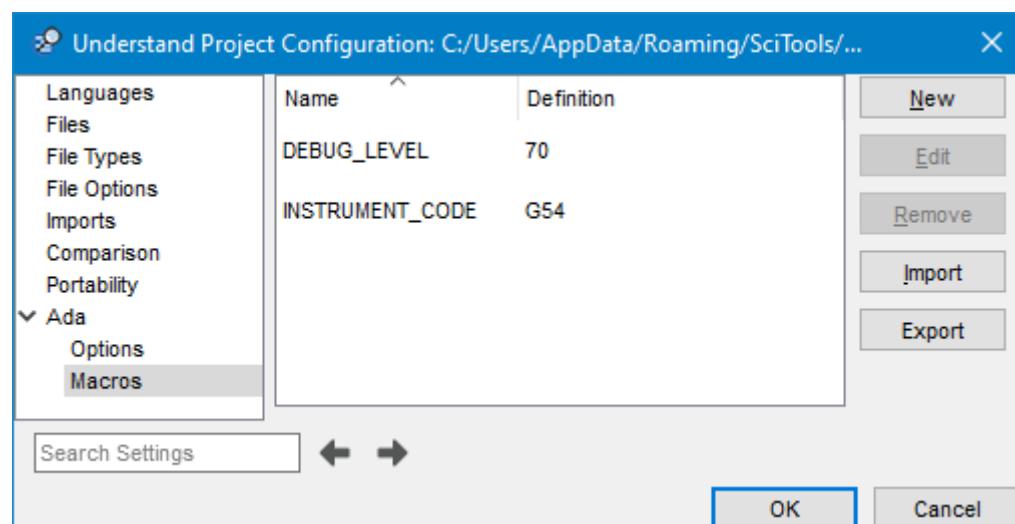
Ada code may contain conditional instructions in pragma statements. For example:

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

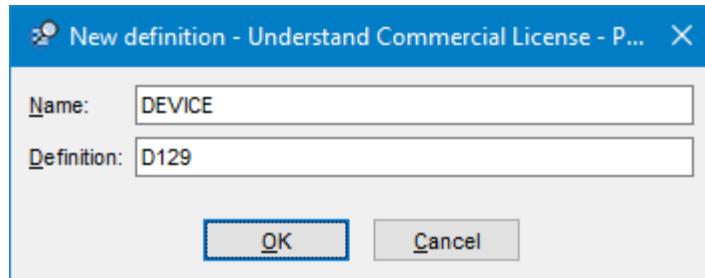
The supported pragmas are IF, IFDEF, ELSIF, ELSE, and ENDIF. These pragmas are similar to preprocessor directives such as #ifdef in C code.

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 91 and the [SciTools website](#).

In the **Ada > Macros** category, 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.

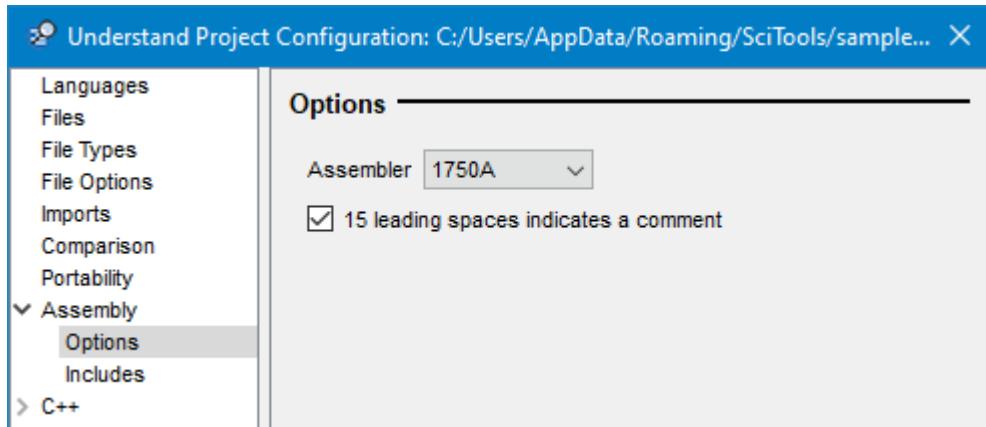
To change the definition of an existing macro without changing the name, select the macro and click **Edit**.

You can import or export a list of macros and their optional definitions by clicking **Import** or **Export** 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.

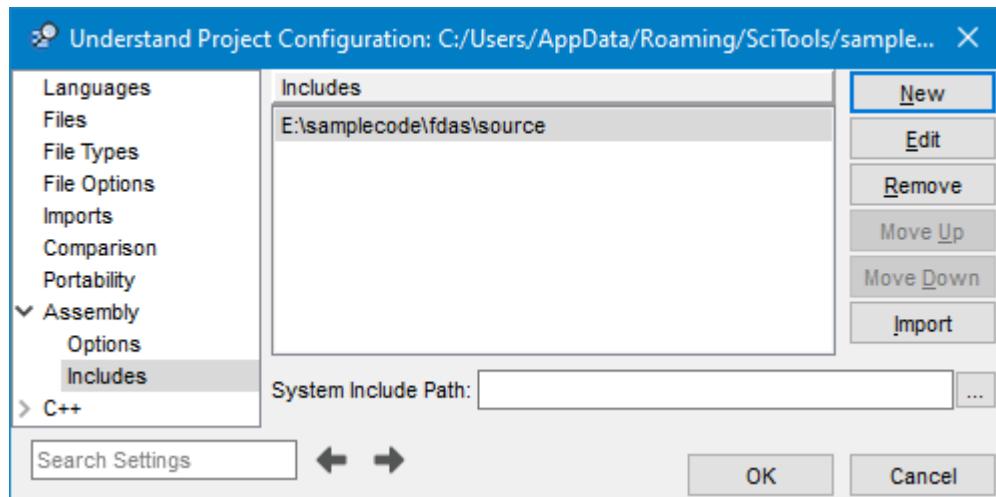


The assemblers supported are:

- Freescale Coldfire 68K
- JIPSE MIL-STD-1750A
- IBM 390

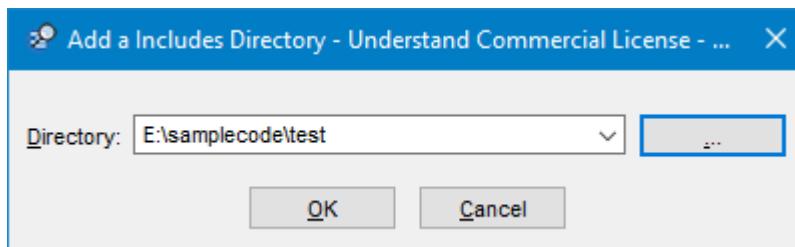
If you select the JIPSE MIL-STD-1750A (1750A) assembler, you can also specify that a line with 15 leading spaces is a comment.

The **Assembly > Includes** category in the Project Configuration dialog allows you 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. 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 files found in system include directories are not added.

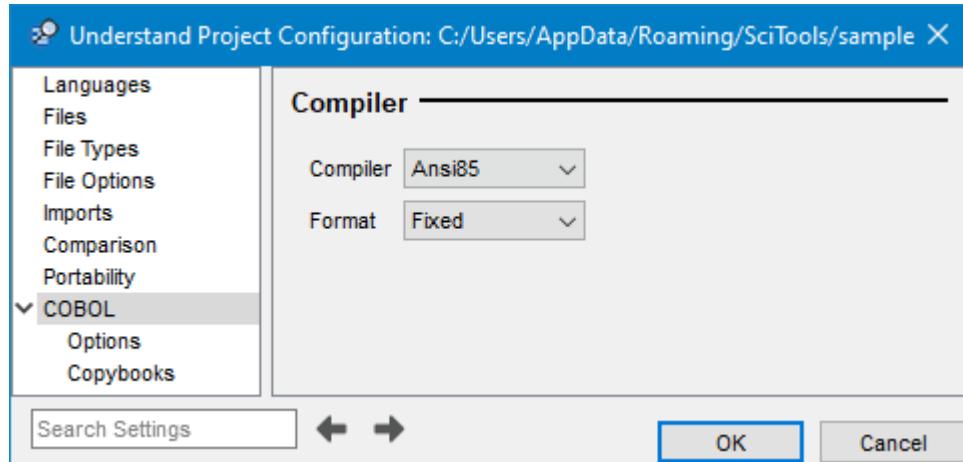
Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

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 *Portability Options* on page 52).

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.



COBOL installations are often highly customized, and we may need to adapt *Understand* for your codebase. Please contact your distributor or email us at support@scitools.com with the specific details of your installation so that we can ensure the code is analyzed correctly.

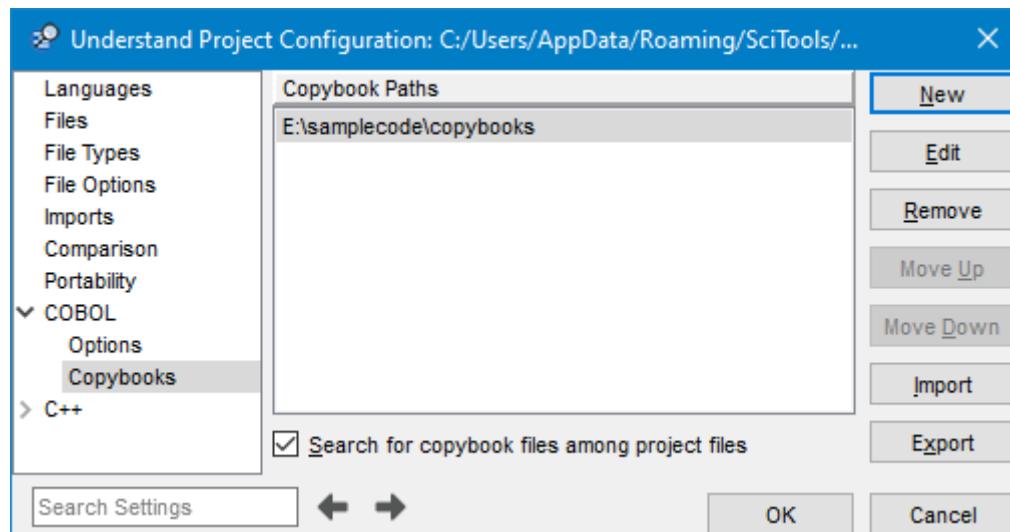
The fields in the **COBOL > Options** category are as follows:

- **Compiler:** Select the compiler that you use. The options are Ansi85, MicroFocus, AcuCobol, IBM, HP OpenVMS, and Unisys.
- **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 *Portability Options* on page 52).

You can import or export a list of copybook directories from a text file by clicking **Import** or **Export** 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.)

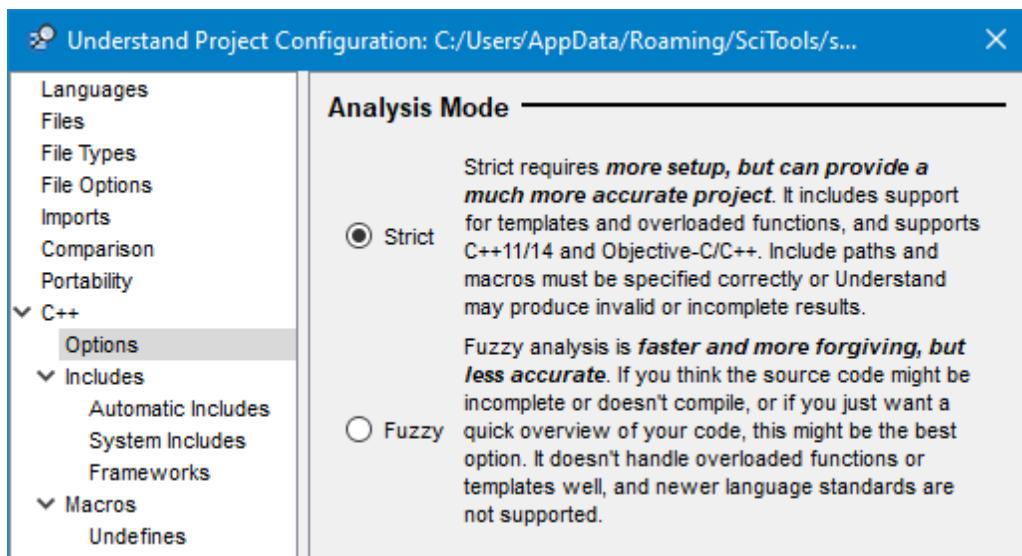
## C++ Options

If your project contains C or C++ code, you should first decide whether to use the “strict” or “fuzzy” analysis mode.

- **Strict mode:** This mode requires more initial setup in *Understand*, but can provide the most accurate results. It includes support for templates and overloaded functions. It supports C++11/14 and Objective C/C++ (for Mac OS and iOS). Include paths and macros must be specified correctly or *Understand* may produce invalid or incomplete results.
- **Fuzzy mode:** This mode makes it faster to create an *Understand* project and is more forgiving, but the results may be less accurate. If your source code may be incomplete or does not compile, or if you just want a quick view of your code as you work on it, this may be the better option. Fuzzy mode does not handle overloaded functions or templates well, and newer language standards are not supported.

For more details, see the [Creating Accurate C/C++ Projects](#) blog post.

To select the mode, choose the **C++ > Options** category in the Project Configuration dialog, and select the mode you want your project to use.



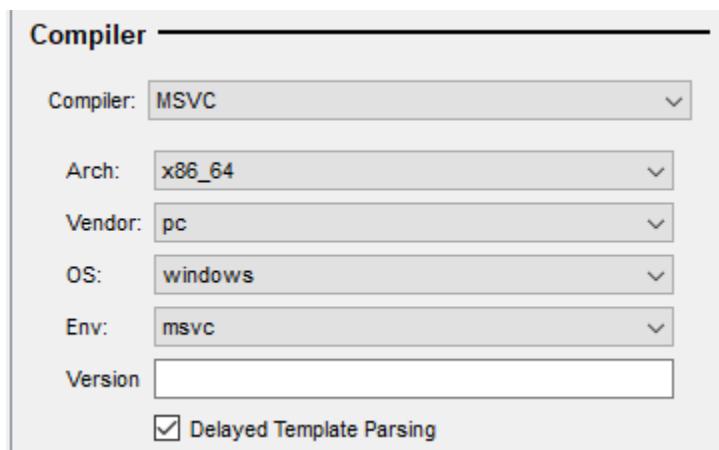
The rest of the options change depending on the mode you select.

## Strict C/C++ Mode Options

If you are using the **Strict** mode in the **C++ > Options** category of the Project Configuration dialog, you can set options in the following categories:

- Compiler
- Language Standard
- Optimization
- Objective-C
- Resolve

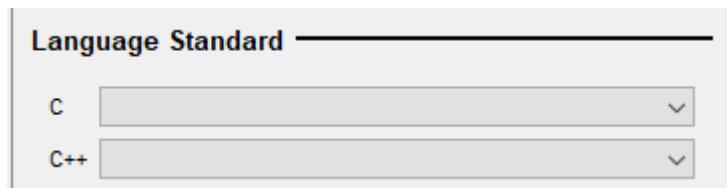
The **Compiler** section of this dialog matches the platform on which you are running *Understand* by default. These fields are used to control which extensions (such as preprocessor defines, header search paths, and language syntax) are analyzed. If your choices here do not match the code used, errors 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 **Compiler** options are as follows:



- **Compiler:** Select the compiler you use. If your compiler is not listed, choose a compiler with similar behavior. The Clang, GCC, and MSVC options provide the following additional fields to provide information about your target.
- **Arch:** Select the architecture of the chip for which your project is written. Examples of the many supported options include ARM, PowerPC64, and x86\_64. Use the “Unknown” option to select the most generic C/C++ code analysis.
- **Vendor:** Select the source of the chip architecture. Examples include 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.

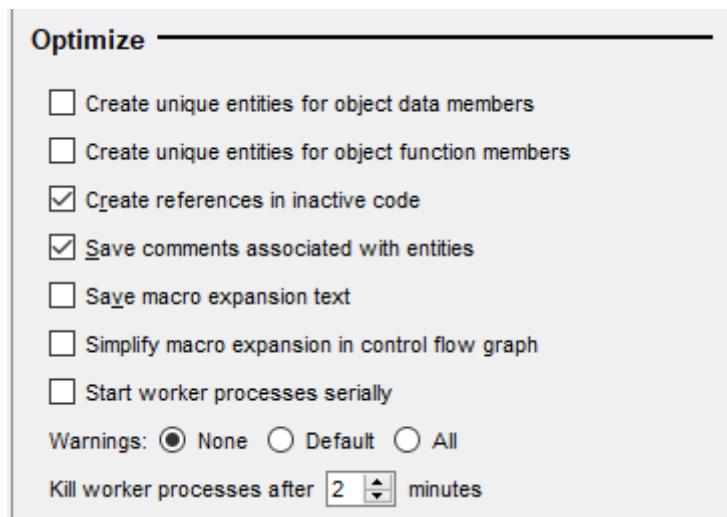
- **Version:** For some Compiler or OS settings, you can also specify the appropriate version number. If your OS is IOS or Mac OSX, specify the operating system version number. If your OS is Windows, specify the Microsoft C (MSC) version of your compiler. For example, specify 1300 for Visual C++ .NET and 1700 for Visual C++ 2012.
- **Delayed Template Parsing:** If your OS is Windows, 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.

The **Language Standard** options are as follows:



- **C Language Standard:** Select the C standard to use when analyzing your C code.
- **C++ Language Standard:** Select the C++ standard to use when analyzing your C++ code.

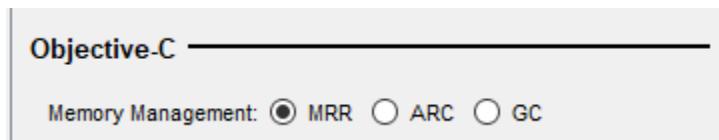
The **Optimize** options are as follows:



- **Create unique entities for object data members:** Check this box if you want separate entities created for each unique instance of data members of a C++ class. The default is to create a single entity for each data member.
- **Create unique entities for object function members:** Check this box if you want separate entities created for each unique instance of function members of a C++ class. The default is to create a single entity for each function member.

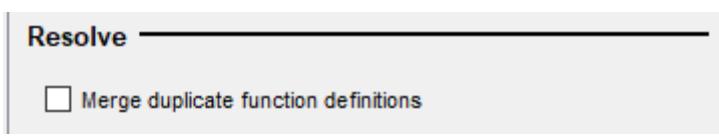
- **Create references in inactive code:** If you wish to exclude cross-reference information for code that is IFDEFed out by the current macro settings, turn this option off. By default, this option is on and cross-reference information for inactive code is included.
- **Save comments associated with entities:** Check box if source code comments before and after an entity should be associated with that entity. On by default.
- **Save macro expansion text:** Check this box if you want to be able to right-click on a macro and choose **Interactive Reports > Expand Macro** from the context menu to see how the macro expands.
- **Simplify macro expansion in control flow graph:** Check this box to avoid creating nodes for control flow structures that are expanded from macro definitions.
- **Start worker processes serially:** The strict analyzer launches a separate worker process for each source file it analyzes. By default, worker processes run in parallel. Check this option if worker processes are not completing to have them run serially instead.
- **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.
- **Kill worker processes after \_\_ minutes:** The strict analyzer launches a separate worker process for each source file it analyzes. If any file takes too long to be analyzed, *Understand* kills the process and no analysis data is generated for that file. *Understand* prints an error message to the analysis log about any files that are not analyzed and continues analyzing the next file. You can reduce the analysis load by, for example, using forward declarations and removing includes. Alternately, you can use this option to increase the analysis time before the process gets killed. By default, worker processes analyze files for up to 2 minutes.

The **Objective-C** options are as follows:



- **Memory Management:** If you use Objective-C, select the memory management mode used. The options are MMR (manual retain-release), ARC (automatic reference counting), and GC (garbage collection).

The **Resolve** options are as follows:



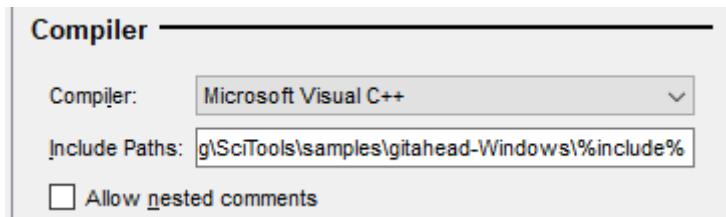
- **Merge duplicate function definitions:** Check this box if the analysis is creating duplicate entities for multiple definitions of the same (not overloaded) function.

**Fuzzy C/C++ Mode Options**

If you are using the **Fuzzy** mode in the **C++ > Options** category of the Project Configuration dialog, you can set options in the following categories:

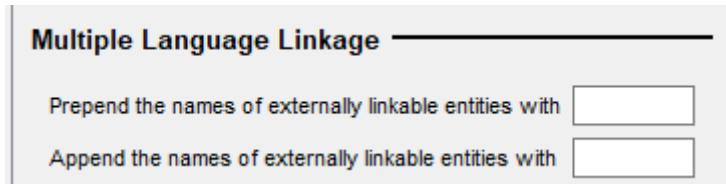
- Compiler
- Multiple Language Linkage
- Optimization
- Resolve

The **Compiler** section of this dialog allows you to select a compiler and include paths for the project analysis. The **Compiler** options are as follows:



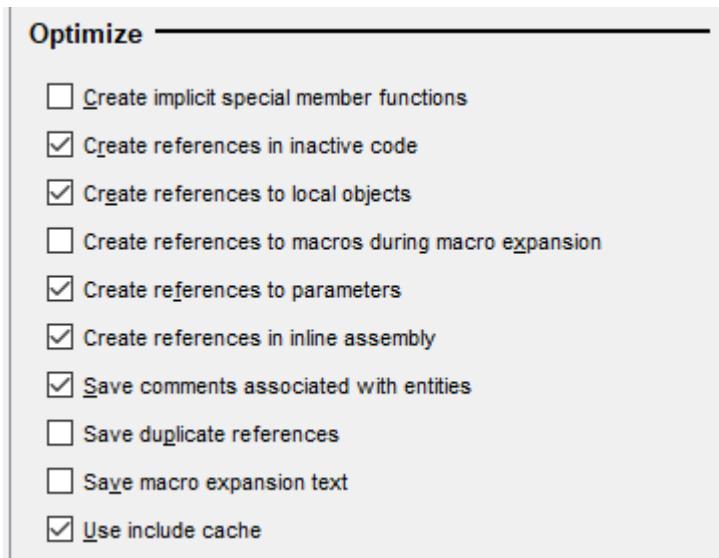
- **Compiler:** Select the compiler/platform that you use. Many different compilers are supported. Your choice affects how *Understand* analyzes the project. Note that not all features of a particular compiler will necessarily be handled.
- **Compiler Include Paths:** Type or paste the path the compiler 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.

The **Multiple Language Linkage** options are as follows:



- **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.

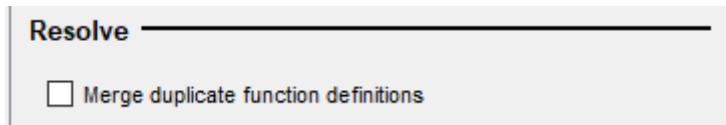
The **Optimize** options are as follows:



- **Create implicit special member functions:** Check this box if you want a default constructor and destructor to be created when the project is analyzed 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 analyzer 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 project analysis. If you wish to exclude variables declared within functions from the analysis, 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 project 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:** Check box if source code comments before and after an entity should be associated with that entity. On by default.

- **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:** Check this box if you want to be able to right-click on a macro and choose **Interactive Reports > Expand Macro** 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.

The **Resolve** options are as follows:



- **Merge duplicate function definitions:** Check this box if the analysis is creating duplicate entities for multiple definitions of the same (not overloaded) function.

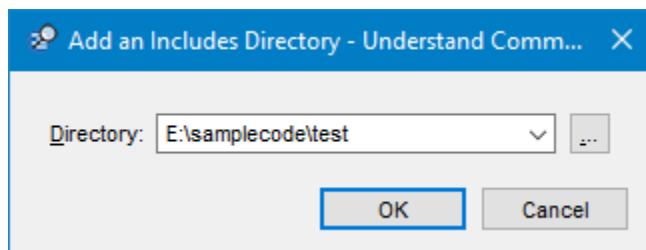
## C++ > Includes Category

The **C++ > 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 90.

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.

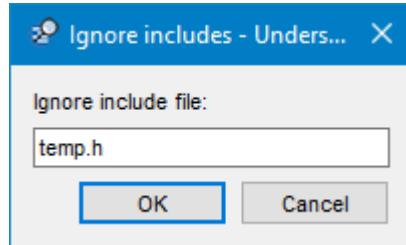
You may use environment variables in include file paths. Use the \$var format on Unix and the %var% format on Windows. You can also use named roots in include file paths (see *Portability Options* on page 52).

You can import or export a list of include directories from a text file by clicking **Import** or **Export** 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++ > Includes** category provides different options depending on whether you are using Strict or Fuzzy mode:

- **Sysroot/SDK:** (Strict mode only) This field allows you to specify the root of the default header search path. For example, if you set Sysroot to /dir, the analyzer searches /dir/usr/include instead of /usr/include. This is useful if you use cross compilers or builds against a different SDK from the host machine. This option corresponds to the --sysroot command line option in compilers such as gcc, icc, and clang. This option is available for all supported platforms.
- **Add found include files to source list:** (Both modes) 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 on.
- **Add found system include files to source list:** (Both modes) 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:** (Both modes) This option directs the analyzer to look among project files as a last resort for missing include files. The default is on.
- **Treat system includes as user includes:** (Fuzzy mode only) This option tells the analyzer to look for system includes (surrounded by <>) using the same strategies as normal includes (surrounded by quotes). If this item is off, the analyzer looks for system includes only in directories defined by the compiler configuration. The default is on.
- **Ignore directories in include names:** (Both modes) 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.
- **Compare files by content instead of path:** (Strict mode only) Check this option if you want include files to be compared by their contents rather than by their file path. The default is off.
- **Analyze unincluded headers in isolation:** (Strict mode only) Check this option to omit analysis of header files that are not included by C/C++ files in the project.

<b>C++ &gt; Includes &gt; Automatic Includes Category</b>	(Both Strict and Fuzzy mode) In the <b>C++ &gt; Includes &gt; Automatic Includes</b> category you can specify include files that should be included before each file in a project. To add a file, click <b>New</b> and browse for the file(s). Then click <b>Open</b> . You can import or export a list of auto include files from a text file by clicking <b>Import</b> or <b>Export</b> and selecting the text file that contains one file path per line. Use <b>Move Up</b> and <b>Move Down</b> to change the order in which files are included.
<b>C++ &gt; Includes &gt; System Includes Category</b>	(Strict mode only) In the <b>C++ &gt; Includes &gt; System Includes</b> category lets you specify system paths that the source code uses. To search a directory and its subdirectories for a framework, click <b>Search</b> . This opens the Missing Header Files tool, which is described in <i>Using the Missing Header Files Tool</i> on page 90. To add a specific location, click <b>New</b> and browse for the folder. Then click <b>Select Folder</b> and then <b>OK</b> . You can import or export a list of framework folders from a text file by clicking <b>Import</b> or <b>Export</b> and selecting the text file that contains one path per line. Use <b>Move Up</b> and <b>Move Down</b> to change the order in which folders are processed.
<b>C++ &gt; Includes &gt; Frameworks Category</b>	(Strict mode only) In the <b>C++ &gt; Includes &gt; Frameworks</b> category lets you specify Mac OS and iOS framework paths that the project uses. To search a directory and its subdirectories for a framework, click <b>Search</b> . Click <b>OK</b> to add the found directories to the list of frameworks. To add a location, click <b>New</b> and browse for the folder. Then click <b>Select Folder</b> and then <b>OK</b> . You can import or export a list of framework folders from a text file by clicking <b>Import</b> or <b>Export</b> and selecting the text file that contains one path per line. Use <b>Move Up</b> and <b>Move Down</b> to change the order in which folders are processed.
<b>C++ &gt; Includes &gt; Ignore Category</b>	(Fuzzy mode only) In the <b>C++ &gt; Includes &gt; Ignore</b> category you can specify individual include files that you wish to ignore during analysis. To add a file to be ignored, click <b>New</b> and type the filename of the include file. Then click <b>OK</b> . The filename can use wildcards, such as moduleZ_*.h, to match multiple files.



Any missing files you choose to ignore when prompted during analysis are added here.

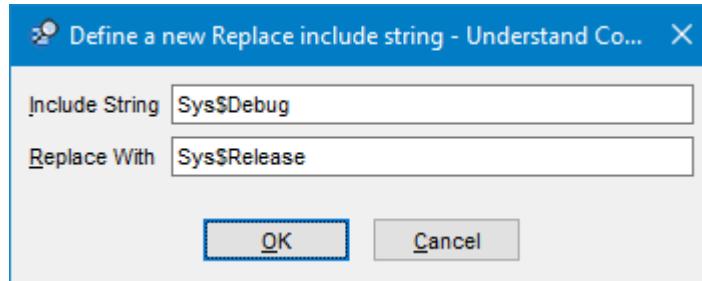
You can import or export a list of files to ignore from a text file by clicking **Import** or **Export** and selecting the text file that contains one filename per line.

**C++ > Includes >  
Replacement Text**

(Fuzzy mode only) 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 or export a list of include strings and their replacements from a text file by clicking **Import** or **Export** 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 **Move Up** and **Move Down** to change the order in which replacements are made.

**C++ > Macros  
Category**

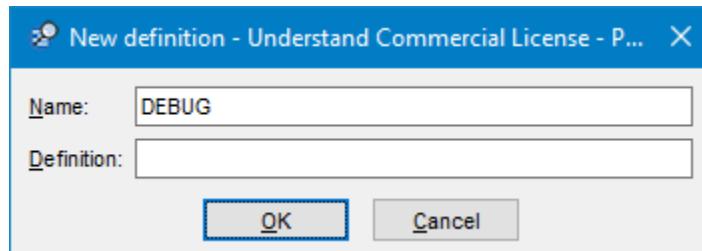
C source code is often sprinkled with pre-processor directives providing instructions and options to the C compiler. Directives such as the following affect what the software does and how it should be analyzed:

```
#define INSTRUMENT_CODE
#ifndef 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).

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.

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 91 and the [SciTools website](#).

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

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

You can import or export a list of macros and their optional definitions from a text file by clicking **Import** or **Export** 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`.

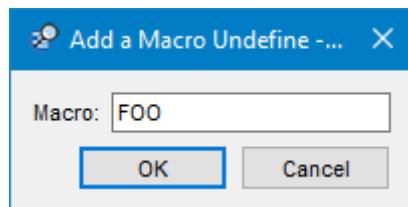
The priority for macro definitions is as follows, from lowest to highest priority:

- 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 command line using `-define`
- 7 Define in source file (`#define` / `#undefine` in source)

---

#### C++ > Macros > Undefines Category

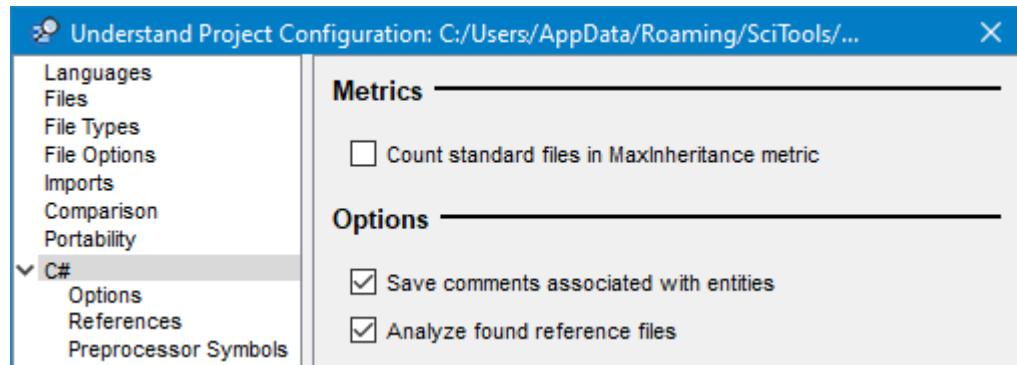
You can list undefined macros in the **C++ > Macros > Undefines** category in the Project Configuration dialog. Click **New** and type the name of a macro that you do not want to be defined. Then click **OK**.



You can import or export a list of undefined macros from a text file by clicking **Import** or **Export** 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# Options

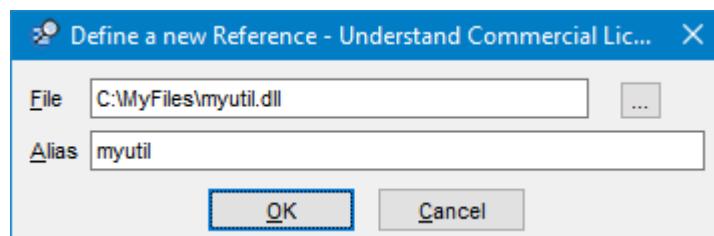
In the **C# > Options** category of the Project Configuration dialog, you can control how C# source code is analyzed. You see this window when you choose the **Project > Configure Project** menu item and select the **C#** category.



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

- **Count standard files in MaxInheritance metric:** Determines whether extending standard classes (those provided with the compiler) are counted when determining the maximum inheritance level. By default, such classes are not counted.
- **Save comments associated with entries:** Choose whether source code comments that occur before and after an entity should be associated with that entity. The default is on.
- **Analyze found reference files:** If this box is unchecked, DLL file contents are not loaded into the project analysis. As a result, classes referenced from DLL files will appear as “unresolved type” entities. The default is on.

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**.



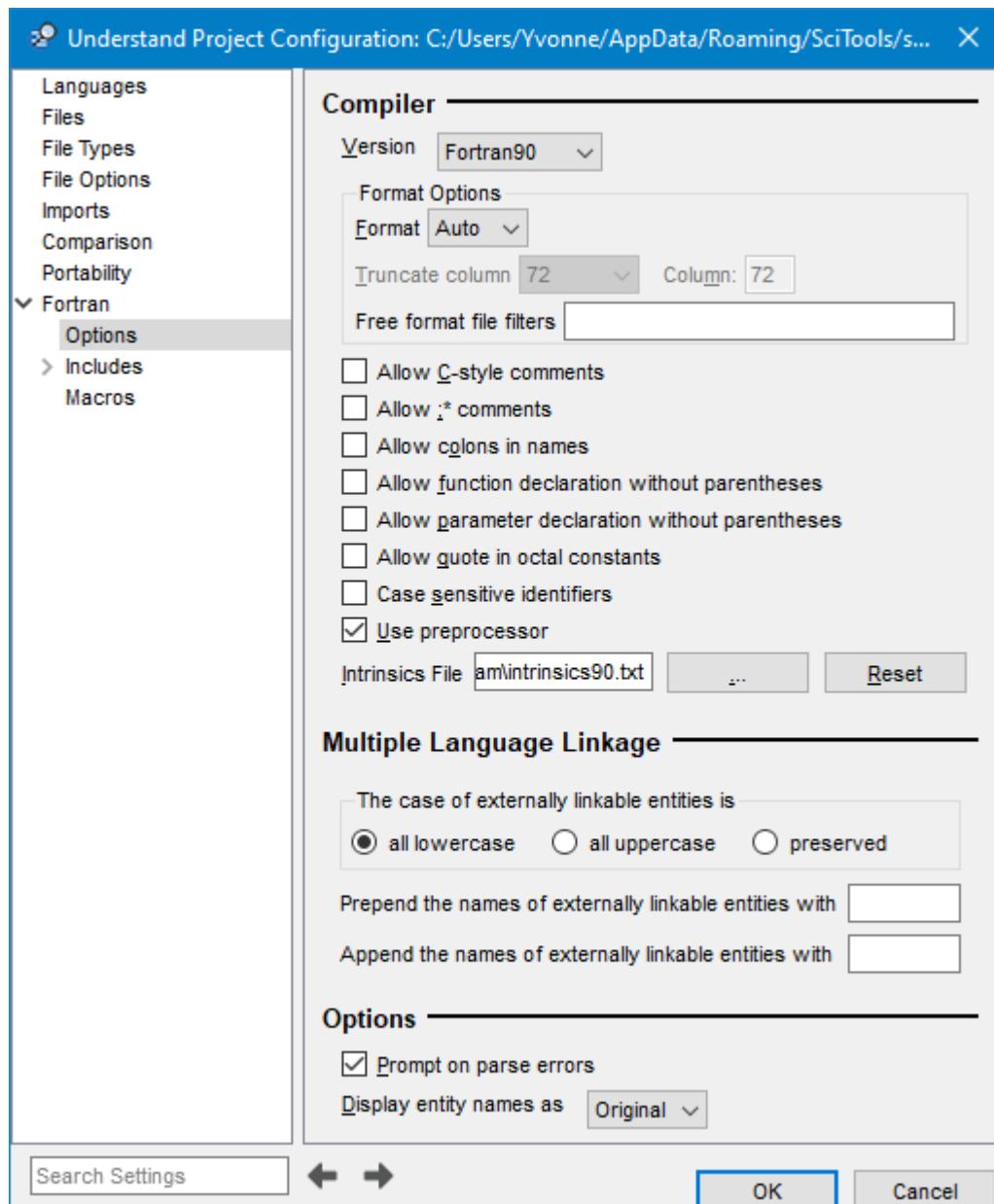
You can import or export a list of reference files and their aliases from a text file by clicking **Import** or **Export** and selecting a file that contains one reference and its alias per line. The file should separate the reference file and its alias with an equal sign (=).

By default, referenced DLL 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 the **C# > Options** category.

In the **C# > Preprocessor Symbols** category, you can click **New** to add symbol names that should be treated as defined when analyzing preprocessor directives such as #if.

## 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.



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, Fortran2003, and

Fortran2008. If you have a mix of code, choose the newest language variant. That is, if you have F77 and F95 code, choose F95.

- **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,” 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.
- **Free format file filters:** Forces files with the specified endings to be interpreted as free format files. Use this option if a project contains both fixed and free format source files and the automatic format detection has trouble identifying free format files. If all free format files end in ".f90", you can add \*.f90 to the free format file filter to make sure they are parsed using free format.
- **Allow C-style comments:** Check this option if your Fortran code contains comments of the form /\* ... \*/.
- **Allow ;\* comments:** Allow the use of end-of-line comments that begin with ;\*.
- **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 parameter declaration without parentheses:** Check this box if you want to allow parameters 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 with intrinsic functions you want analyzed. Default files (intrinsics77.txt, intrinsics90.txt, and intrinsics95.txt) are provided in *Understand's <install\_directory>/conf/understand/fortran* directory.
- **Case of externally linkable entities:** Choose which case should be used for entities that are “exported to” or “imported from” other languages. For example, if an entity is declared in Fortran as “MYITEM” and you choose “all lowercase” here, other languages would be expected to use “myitem” to call that entity. Likewise,

Fortran source code could call a function as “MYPROG” that is defined in C as “myprog” if you choose “all lowercase” here.

- **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, a prompt asks how to handle any parsing errors. 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 90 and the [SciTools Support website](#).

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 select the directory, click **Select Folder**, 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 files as source files in the **Files** category.

You can import or export a list of include directories from a text file by clicking **Import** or **Export** and selecting the file. The file must contain one directory path per line. (In 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 68.

---

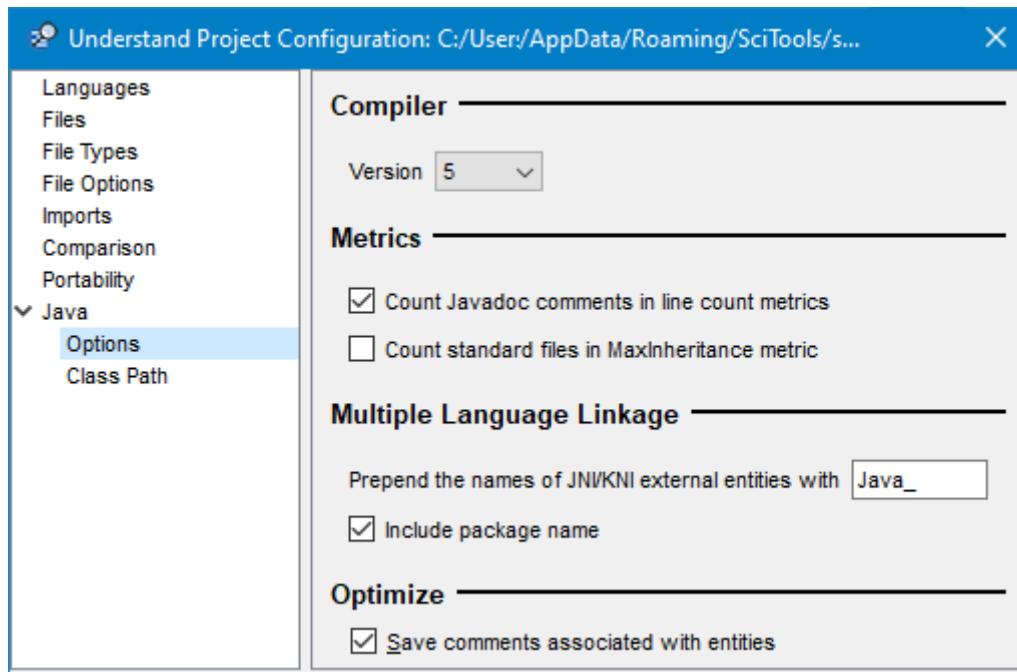
## Other Fortran Categories

For information about the **Fortran > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on page 71.

For information about the **Fortran > Macros** category, see *C++ > Macros Category* on page 71. The following predefined macros are supported: \_\_LINE\_\_, \_\_FILE\_\_, \_\_DATE\_\_, and \_\_TIME\_\_.

## 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.



- **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, 6, 7, 8, 9, and 10-15.
- **Count Javadoc comments in line count metrics:** If this box is checked, Javadoc comments are included when computing the CountLine, CountLineComment, and RatioCommentToCode metrics. The default is on.
- **Count standard files in MaxInheritance metric:** Determines whether extending standard classes (those provided with Java) are counted when determining the maximum inheritance level. By default, such classes are not counted.
- **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:** Check box if source code comments before and after an entity should be associated with that entity. On by default.

## Java > Class Path Category

The **Java > Class Path** 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 and class files and can be contained in files with .jar, .jmod (which also may include header files), and .zip file extensions.
- 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. Then click **OK**.



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

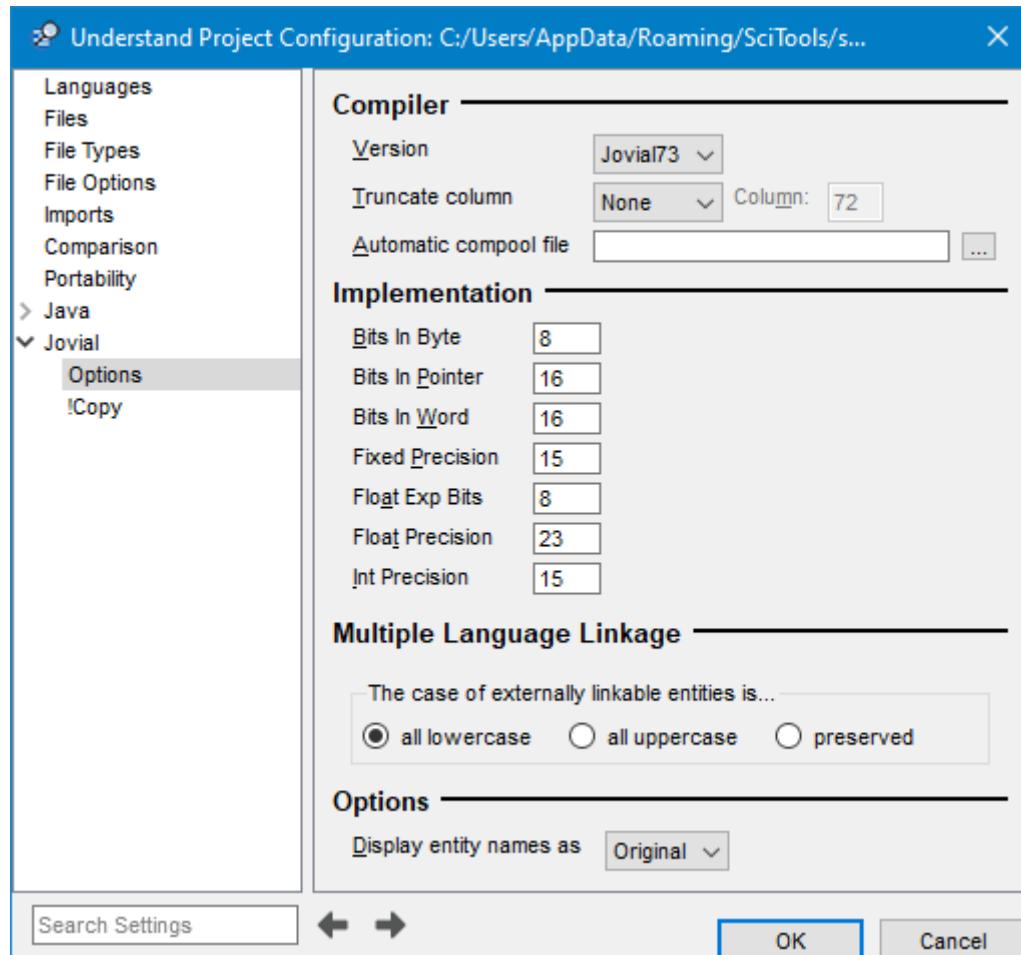
- 1 Click **New Jar**.
- 2 Locate and select .jar, .jmod, or .zip file(s). You can select multiple files while holding down the Ctrl key. You can provide a relative path to a file by typing the path directly in the Jar File field rather than browsing for a file. Then click **Open**.

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 or export a list of class paths and/or jar files from a text file by clicking **Import** or **Export** 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.



JOVIAL installations are often highly customized, and we may need to adapt *Understand* for your codebase. Please contact your distributor or email us at support@scitools.com with the specific details of your installation so that we can ensure the code is analyzed correctly.

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.
- **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”.
- **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.

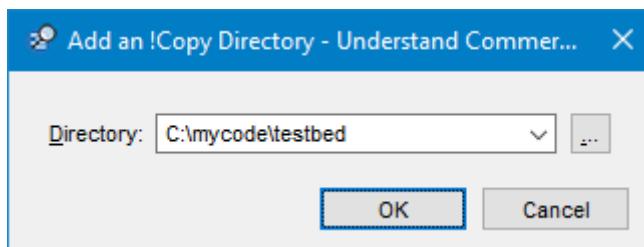
---

#### 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**.



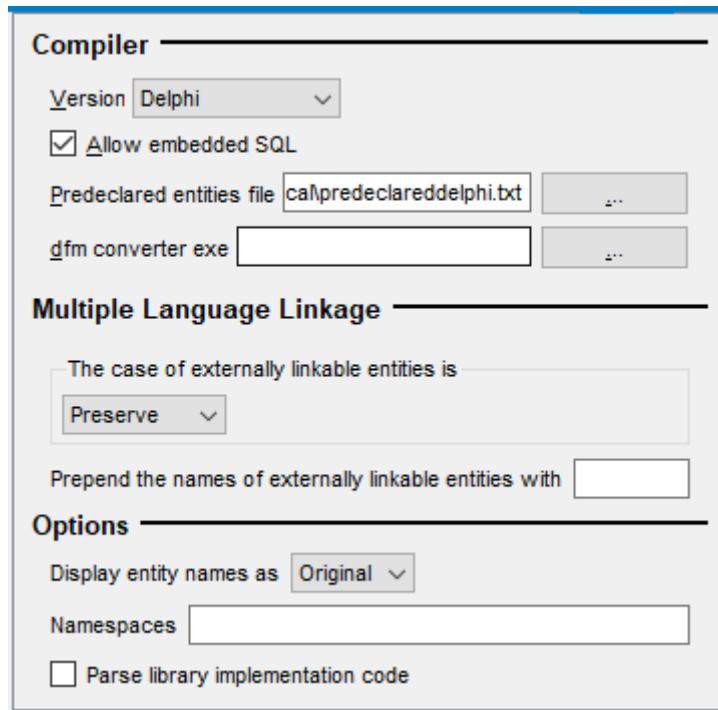
- 2 Click the ... button and browse to the directory you want to add.
- 3 Click **OK**.

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 or export a list of directories to be searched for files named in !COPY directives from a text file by clicking **Import** or **Export** 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.



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 Delphi, Compaq/DEC/HP, Pascal86, and Turbo. Select Compaq for legacy DEC Pascal projects. *Understand* supports all versions of Embarcadero's Delphi language and Embarcadero's Turbo Pascal language. It also supports ISO 7185:1990 (also known as Unextended Pascal) with HP Pascal extensions.
- **Allow embedded SQL:** Check this box to enable parsing of embedded SQL statements in your source code. Ingres embedded SQL statements are supported.
- **Predeclared entities file:** Click ... to select a text file (\*.txt) containing predeclared routines, types, constants, and parameters used in your source code. Two versions of this file are provided in <install\_dir>/conf/understand/pascal: predeclared.txt and predeclareddelphi.txt. The default is set based on 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 analyzed 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 entities that are "exported to" or "imported from" other languages. For example, if an entity is declared in Pascal as "MYITEM" and you choose "all lowercase" here, other

languages would be expected to use “myitem” to call that entity. Likewise, Pascal source code could call a function as “MYPROG” that is defined in C as “myprog” if you choose “all lowercase” here.

- **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.
- **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.
- **Namespaces:** List the namespaces that are automatically “used” by all units in the project. The names must be separated by spaces, blanks, or semicolons.
- **Parse library implementation code:** Check this box if you want to parse implementation parts of the Delphi standard library files. Off by default. See *Pascal > Standard Library Paths Category* on page 82 for more about standard libraries.

---

#### 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 this category, see *C++ > Macros Category* on page 71.

---

#### 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 named 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 Click **Move Up** or **Move Down** to change the order in which libraries are checked.

You can import or export a list of directories to be searched for standard libraries from a text file by clicking **Import** or **Export**. 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:

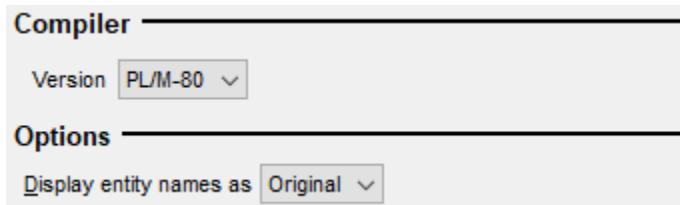
- 1 Click the **New** button. Then click the ... button and browse to a directory. Click **OK**.
- 2 Click **Move Up** or **Move Down** to change the order in which paths are checked.

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

You can import or export a list of directories to search from a text file by clicking **Import** or **Export** 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.



PL/M installations are often highly customized, and we may need to adapt *Understand* for your codebase. Please contact your distributor or email us at support@scitools.com with the details of your installation so that we can ensure the code is analyzed correctly.

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 90 and the [SciTools Support website](#).

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, include directories are searched in the order they appear in the dialog. Click **Move Up** or **Move Down** to change the order in which directories are searched.

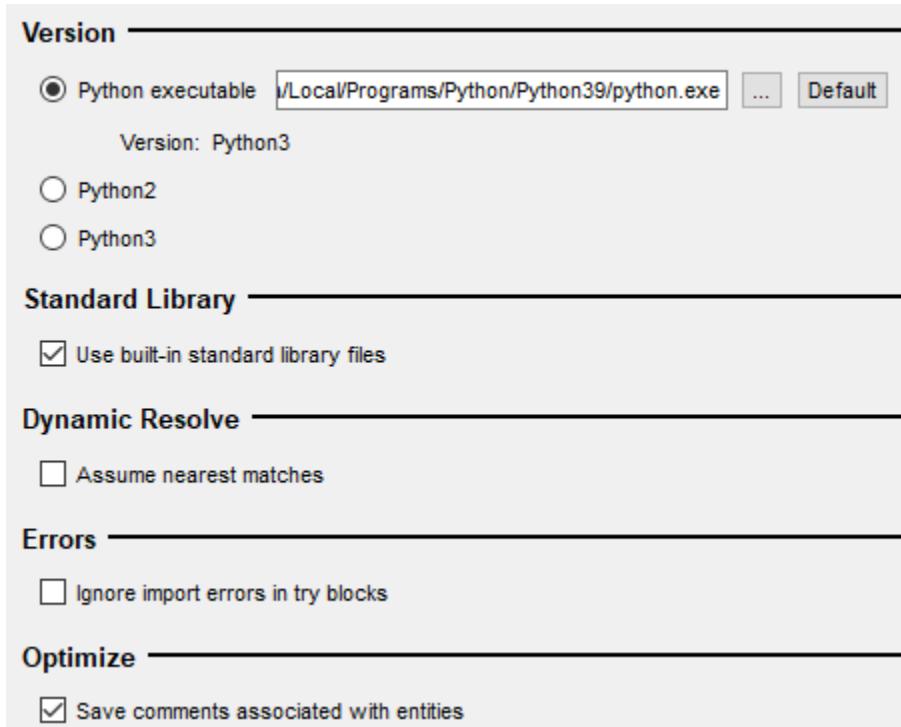
Typically include files not directly related to your project (such as system-level includes) and that you do not want to analyze fully are defined here. For project-level includes you want analyzed, add those include files as source files in the **Files** category.

You can import or export a list of include directories from a text file by clicking **Import** or **Export** 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 68.

For information about the **PL/M > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on page 71.

## 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 **Version** section lets you specify the version of Python to use. You can select one of the following:

- **Python executable.** Click ... and browse for the location of the file you use to run Python programs. The version of this executable will be detected and displayed. In addition, the Python interpreter's `sys.path` variable is examined to find the directories to be searched for modules. The **Default** button fills in a Python executable path if one is found in the `PATH` environment variable definition.
- **Python 2.** Select this option to analyze the project using the Python 2 standard.
- **Python 3.** Select this option to analyze the project using the Python 3 standard.

**Use Built-in Standard Library Files:** By default, *Understand* uses the files in the `./conf/understand/python/python2` or `./conf/understand/python/python3` subdirectory of the *Understand* installation to resolve Python standard library entities. These files contain stubs for such entities. You can disable this part of the search here.

If a module is found in both the built-in libraries (*Understand*'s copy) and the installed Python libraries, the version in the installed libraries takes precedence.

**Assume nearest matches:** Enables resolving attribute references to match attributes defined in the same file or imported files. This option is now off by default.

**Ignore import errors in try blocks:** Suppresses warning messages due to import statements that occur in try-except blocks.

**Save comments associated with entities:** Check box if source code comments before and after an entity should be associated with that entity. On by default.

---

### 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 or export a list of directories from a text file by clicking **Import** or **Export** 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 68.

---

## VHDL Options

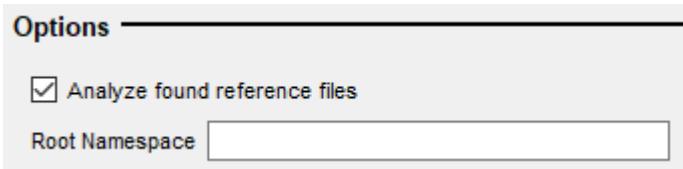
There are currently no Project Configuration options specifically for VHDL.

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.

## Visual Basic (.NET) Options

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



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

- **Analyze found reference files:** If this box is unchecked, DLL file contents are not loaded into the project analysis. As a result, classes referenced from DLL files will appear as “unresolved type” entities. The default is on.
- **Root Namespace:** Specify the root namespace for this project.

In the **Visual Basic > Imported Namespace** category of the Project Configuration dialog, you can click **New** to name a namespace to be imported or **Import** to select a text list file that contains a list of namespaces to import.

In the **Visual Basic > Preprocessor Symbols** category of the Project Configuration dialog, you can click **New** to specify the name and definition of preprocessor symbols that should be treated as defined when analyzing the project. Use **Import** to select a text list file that contains a list of symbols to import.

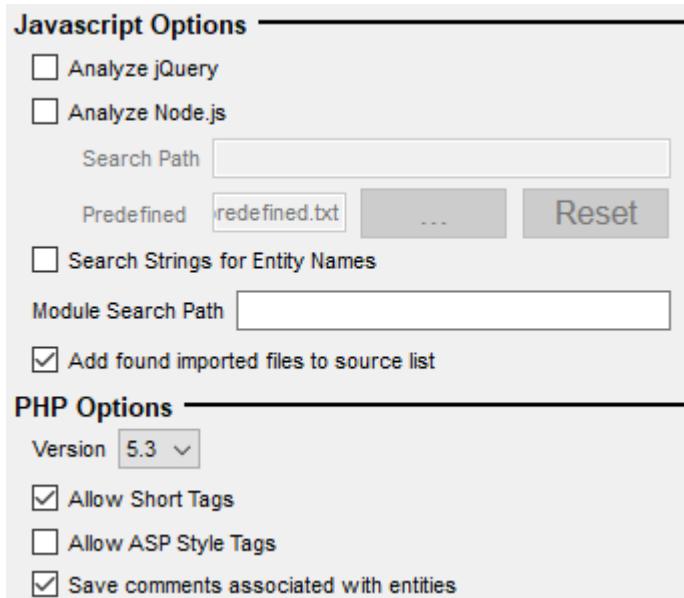
In the **Visual Basic > References** category of the Project Configuration dialog, you can click **New** to name a reference to be imported or **Import** to select a text list file that contains a list of references to import.

## 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, Typescript, and XML. For some file types, such as XML, only line count metrics are generated.

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



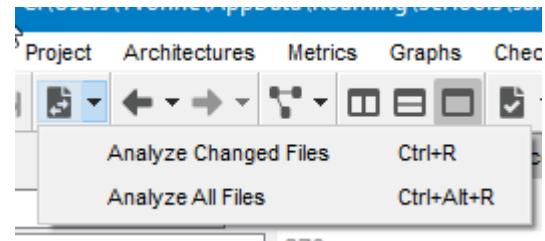
- **Analyze jQuery:** Check this box if you want the analysis to interpret a `$(...)` call as a jQuery call. A **JQuery Selector** entity is created for the string literal parameter passed to `$`. For example, `$(".foo")` would create a JQuery Selector entity named `".foo"`. Also, a **JQuery Selector Uses** category is added to the Information Browser for files and functions that use the jQuery selectors. By default, this option is off.
  - **Analyze Node.js:** Check this box to recognize the Node.js “require” function. A **Requires** category is added to the Information Browser for files that contain calls to “require” and a **Required by** category is added to the Information Browser for files named in a “require” call. Requires and Required By graph views are also available.
- The **Search Path** field allows you to specify a semicolon-separated list of directories to be searched in the order given for files named in “require” calls.
- The **Predefined** field lets you specify a file containing a list of predefined node.js modules. The default is `scitools\conf\understand\javascript\nodejs_predefined.txt`.
- **Search Strings for Entity Names:** Check this box if you want strings within JavaScript code to be searched for references to entities.
  - **Module Search Path:** You may specify a search path for JavaScript modules.
  - **Add found imported files to source list:** Check this box to add imported JavaScript files to the project. On by default.
  - **PHP Version:** Select the version of PHP used by your project. The options are 5.3, 5.4, 5.5, 5.6, 7.0, and 7.1.
  - **Allow Short Tags:** Check this box if your PHP code ever uses the short form of PHP tags. On by default.
  - **Allow ASP Style Tags:** Check this box if your PHP: Hypertext Preprocessor (PHP) code ever uses Active Server Pages (ASP) style tags.
  - **Save comments associated with entities:** Check box if source code comments before and after an entity should be associated with that entity. On by default.

## Analyzing the Code

Once you configure a project, *Understand* can analyze the project. During analysis, the source files are examined and data is stored in a database to allow quick browsing of large projects.

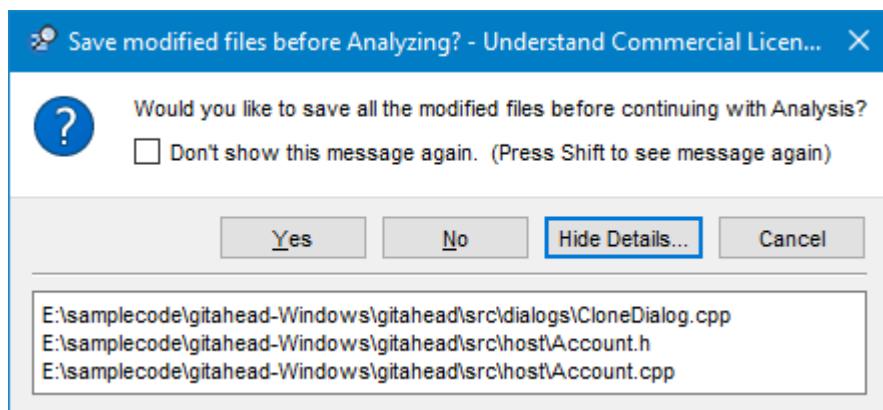
When you create a project or modify a project's configuration, a prompt to analyze the project appears automatically. You can also analyze the project in the following ways:

**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 use the toolbar icon shown here or press 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)

If some files have been modified but not saved, you are asked whether you want to save all the modified files. You can click **Show Details** to see a list of the files that will be saved.



Analyzing a large project can take some time. The status bar shows progress as the project is analyzed in the background.



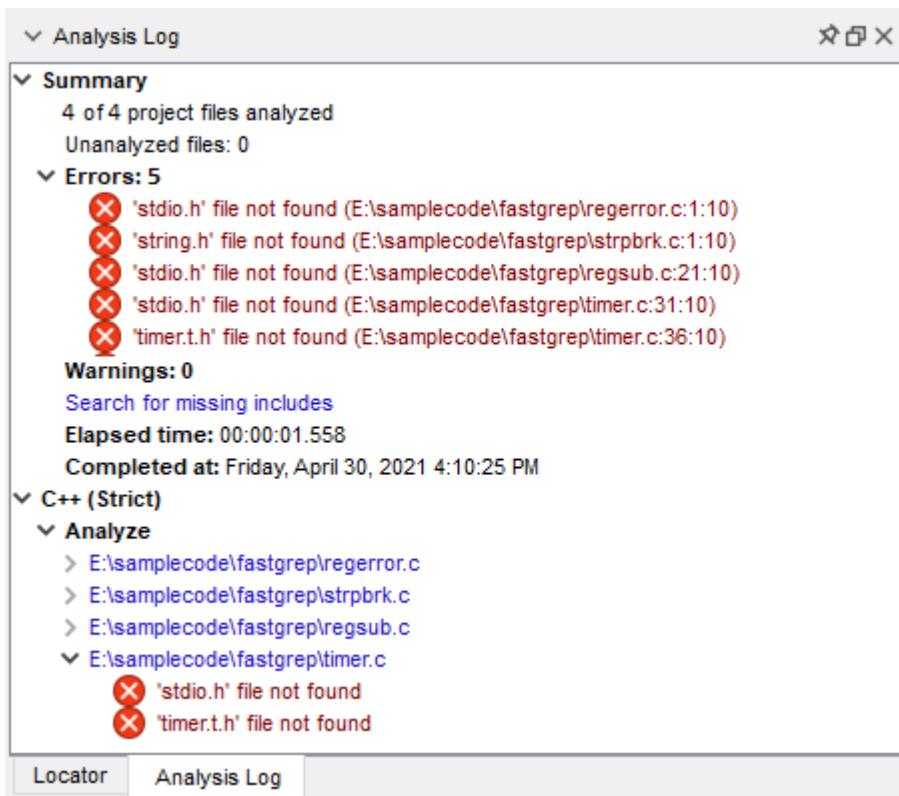
To pause or cancel the project analysis, click the X next to the progress bar. You will be asked if you want to Abort the analysis or Continue the analysis.

If you cancel the analysis, the project will likely be in an incomplete state. It is recommended that you save any unsaved work and re-run the project analysis. In some cases, you may need to restart *Understand*.

If you choose **Project > Analyze All Files** when the Project Overview is open, statistics about the accuracy and errors found when parsing are added to the top of the Project Overview. Click a statistic in the gray bar to open a view to correct the problem.

<b>25</b>	<b>67%</b>	<b>113</b>	<b>3</b>	<b>39,852</b>	<b>103</b>	<b>31</b>	<b>495</b>	<b>184</b>
Missing Includes	Parse Accuracy	Errors	Warnings	Lines	Files	Classes	Functions	Subprograms

You can see the results of the analysis in the Analysis Log. To open the Analysis Log, click the analysis information in the lower-left corner of the *Understand* window. Or, choose **View > Analysis Log**.



Double-click on any error to jump to the source code for that error. If there are missing includes, the Analysis Log contains a link to "Search for missing includes". You can double-click this link to open the Missing Header Files dialog (see page 90).

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

If you have analyzed the project during this session, you can choose **View > Analysis Log** command to reopen the log. See *Analyze Category* on page 103 for options that affect the project analysis.

**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 287 for details on using "*und*".

## Improving the Analysis

If your project analysis results in warnings about missing files, choose **Project > Improve Project Accuracy > Missing Includes** and see page 90 for how to use the Missing Header Files tool.

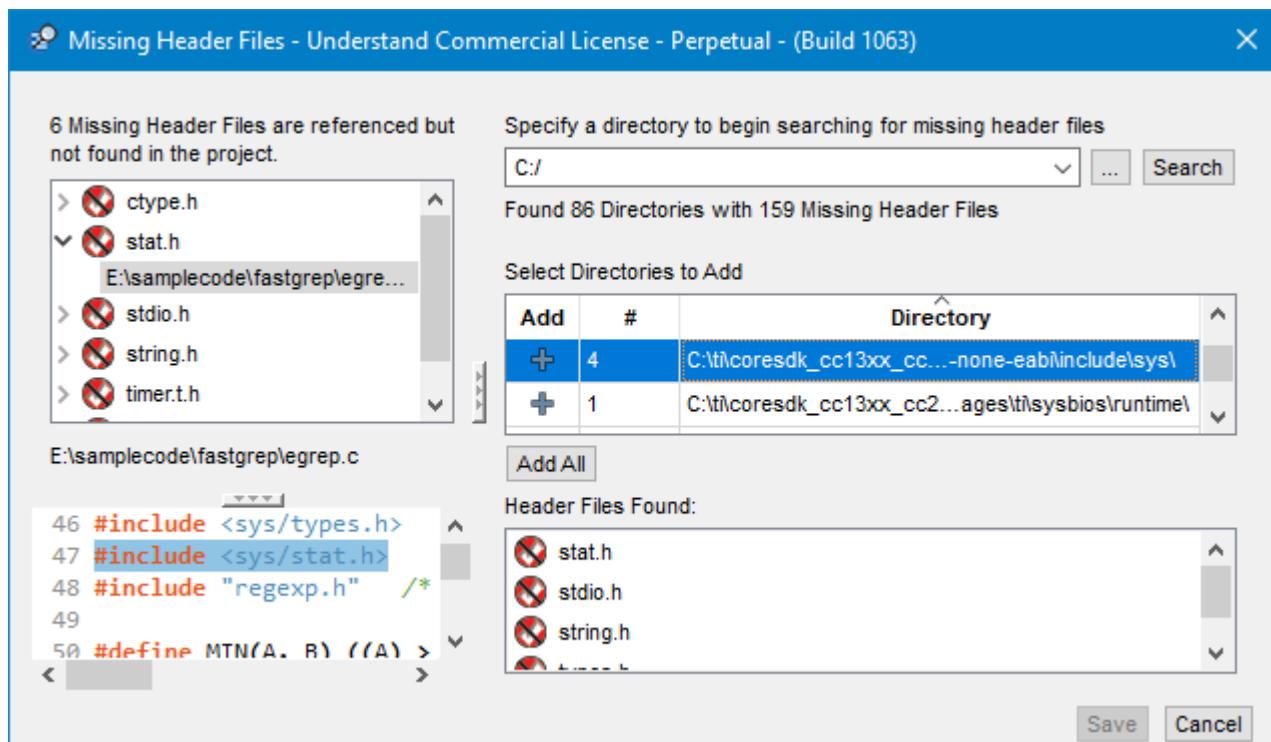
If your project analysis results in warnings about undefined macros, choose **Project > Improve Project Accuracy > Undefined Macros** and see page 91 for how to use the Undefined Macros tool.

For other types of warnings, revisit the categories of the *Project Configuration Dialog* on page 39 to make sure your project is configured correctly. Multiple similar errors can often be fixed quickly. For advice about tuning configuration settings to improve your project analysis, choose **Project > Improve Project Accuracy > More Information**.

## Using the Missing Header Files Tool

Configuring your include file directories is important to improving the accuracy of project analysis. If your project analysis results in warnings about missing files, use the Missing Header Files tool as follows:

- 1 Choose **Project > Improve Project Accuracy > Missing Includes** or double-click the link to “Search for missing includes” in the Analysis Log.
- 2 In the top-left area, expand the item for a missing header file and select the source file path to see references to a header file. You will see the code that includes this missing file in the lower-left area.



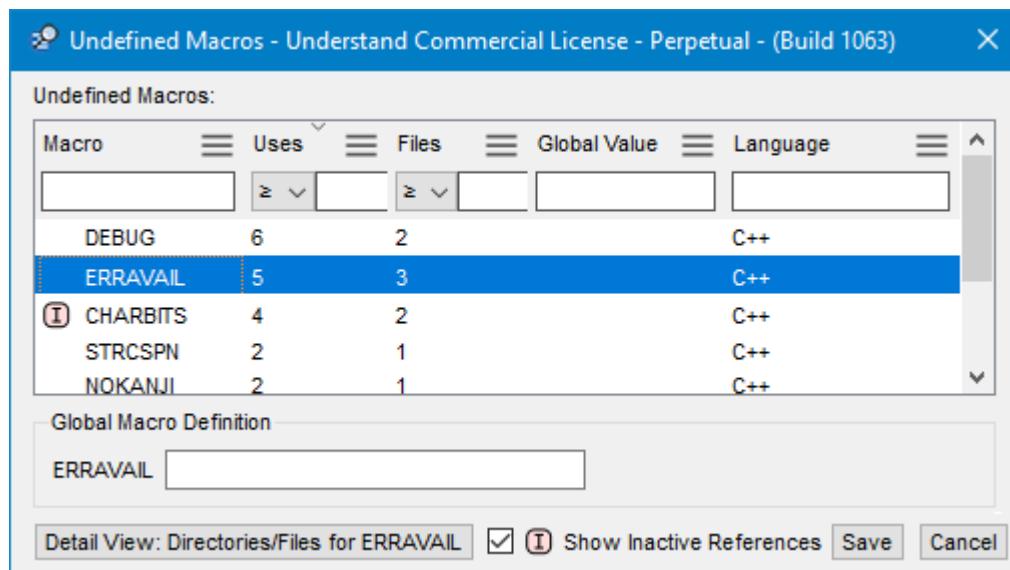
- 3 In the **Specify a directory to begin searching** field, click the ... button and browse to find a directory that may contain one or more missing header files. All subdirectories of the directory you select will be searched. The search is case-insensitive on Windows.

- 4 Click **Select Folder** and then click the **Search** button in the Missing Header Files dialog. If any of the missing files are found, the number of files found and their names are listed in the Select Directories to Add area.
- 5 If any of the header files you want to use in the analysis are found, click the + icon next to a directory you want to add or the **Add All** button below the list.
- 6 The list of missing headers files is updated to show which header files remain missing with a red icon and which files have been found with a green icon. You can continue searching and adding additional directories as needed.
- 7 Click **Save** to apply your changes to the project configuration.
- 8 Click **Yes** at the prompt that asks if you want to analyze the project now.

## Using the Undefined Macros Tool

Configuring your macro definitions is important to improving the accuracy of project analysis. If your project analysis results in warnings about undefined macros, use the Undefined Macros tool as follows:

- 1 Choose **Project > Improve Project Accuracy > Undefined Macros**.



- 2 Select a macro in the list. 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 157 for more about using these filter fields.
- 3 Type a definition for the macro in the **Global Macro Definition** field.
- 4 Click the **Detail View** button to see the code where the selected macro is used. In this view, you can define a macro for a specific file or folder instead of project-wide.
- 5 You can select other macros and type definitions for them before saving changes.
- 6 Click **Save** to apply your changes to the project configuration.
- 7 Click **OK** in the Project Configuration dialog to save the project configuration.
- 8 Click **Yes** at the prompt that asks if you want to analyze the project now.

---

## Chapter 4    Setting Options

This chapter shows how to control the behavior of *Understand*. These settings apply to all projects you work with on your computer.

This chapter contains the following sections:

<b>Section</b>	<b>Page</b>
Understand Options Dialog	93
General Category	94
User Interface Category	96
Key Bindings Category	101
CodeCheck Category	103
Analyze Category	103
Dependency Category	104
Editor Category	105
Graphs Category	114
Annotations Category	118
User Tools Category	120
Python Category	120

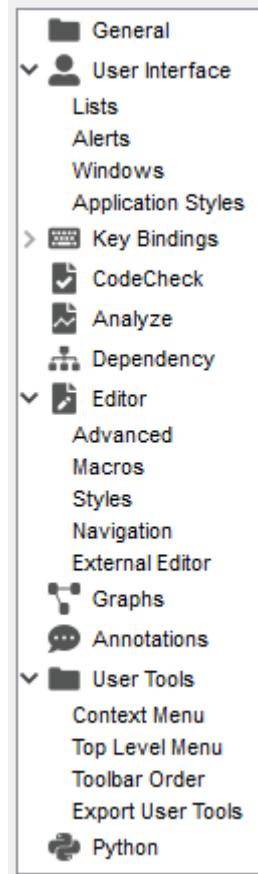
## Understand Options Dialog

*Understand* allows you to control a number of aspects of its operation using the *Understand Options* dialog. To open this dialog, choose **Tools > Options**. (On MacOS, the command is **Understand > Preferences** due to OS conventions.) This dialog provides options to set in the categories shown to the right:

The subsections that follow describe each of the categories:

- *General Category* on page 94
- *User Interface Category* on page 96
- *Key Bindings Category* on page 101
- *Analyze Category* on page 103
- *Dependency Category* on page 104
- *Editor Category* on page 105
- *Graphs Category* on page 114
- *Annotations Category* on page 118
- *User Tools Category* on page 120
- *Python Category* on page 120

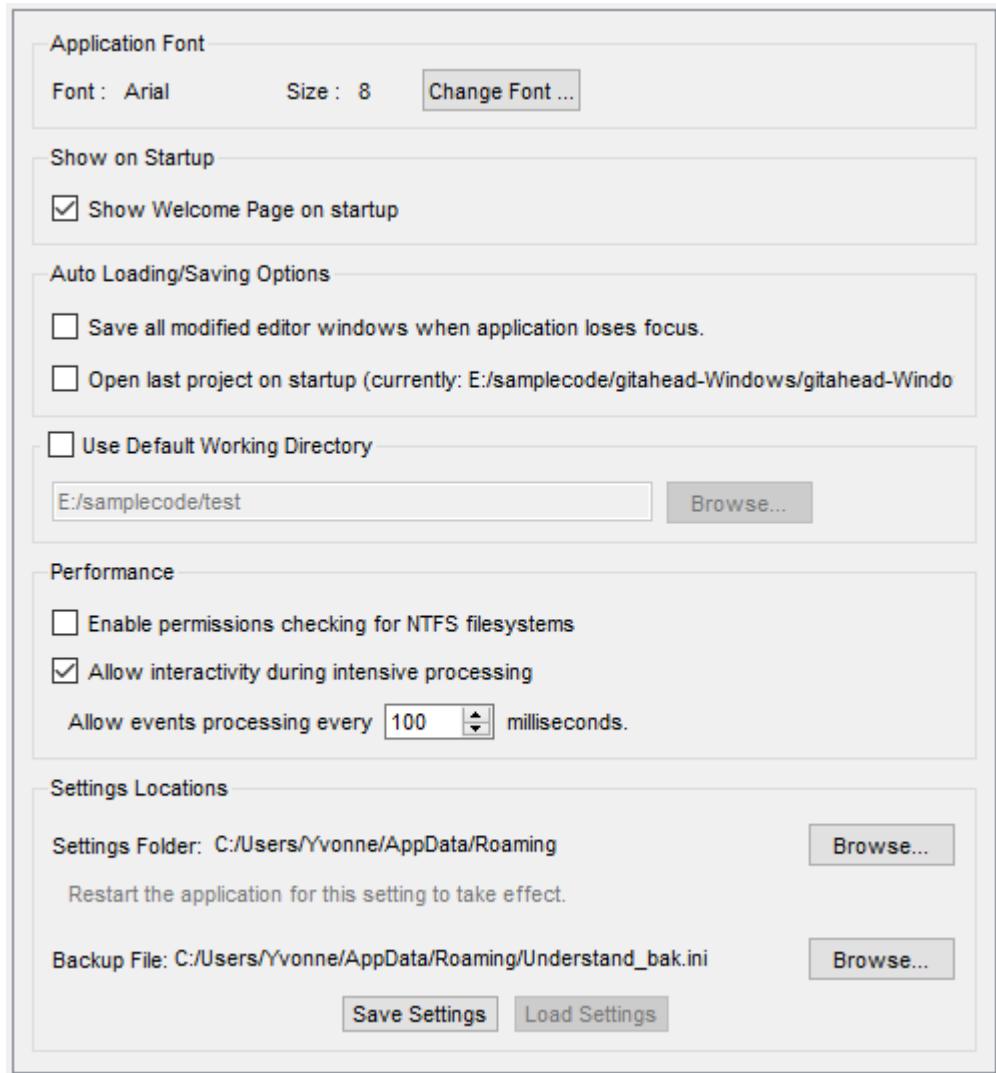
You can use the **Search Settings** box to find options. For example, type “color” and click the right arrow to move through categories with options related to colors.



Click **Restore Defaults** to restore the default settings. You will be asked if you want to restore all settings or just the settings for the currently selected category.

## 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 Welcome page on startup:** If checked (on by default), the Welcome tab (see page 19) 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.
- **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 improve background processing performance by reducing this value.
- **Settings Folder:** Specify 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.
- **Backup File:** Specify a location for the backup *Understand* initialization file. By default, this file is the *Understand\_bak.ini* file in the Settings Folder. You can click **Save Settings** to save all the current option settings to this file. If you want to load different settings, change the Backup File location to point to a different or edited file and click **Load Settings**.

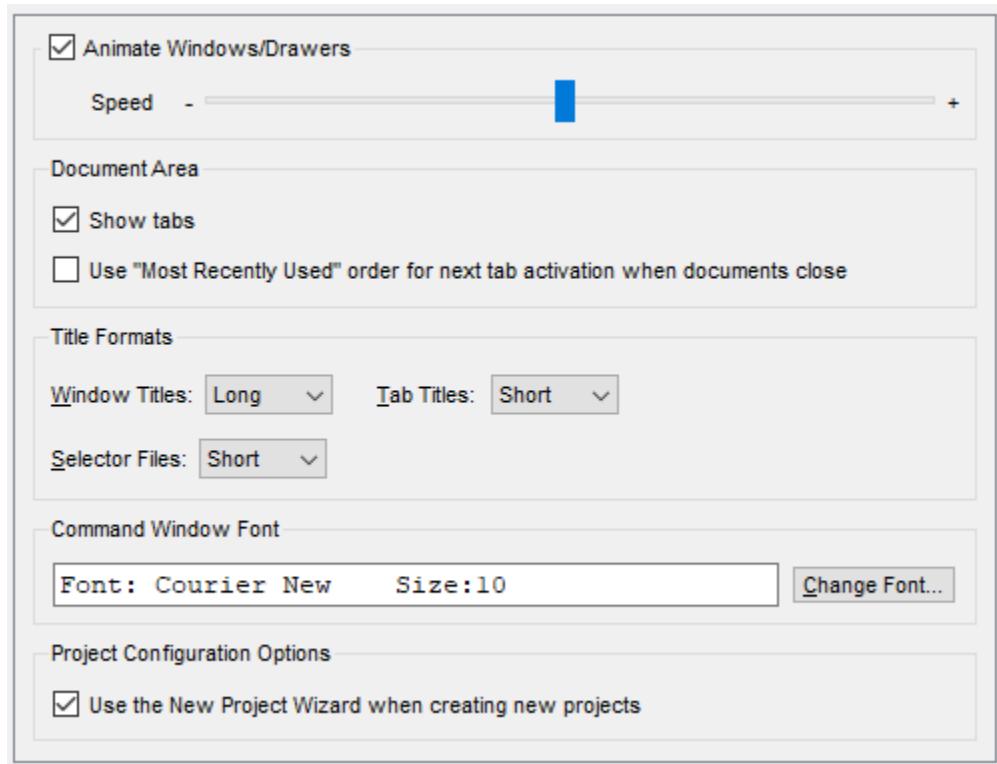
Note that *Understand* supports Dark Mode by following your system settings. If you switch between normal and dark modes, restart *Understand* to have it switch to your new mode.

## User Interface Category

The **User Interface** category has general options and options in the following subcategories:

- User Interface > Lists Category on page 97
- User Interface > Alerts Category on page 98
- User Interface > Windows Category on page 99
- User Interface > Application Styles Category on page 100

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

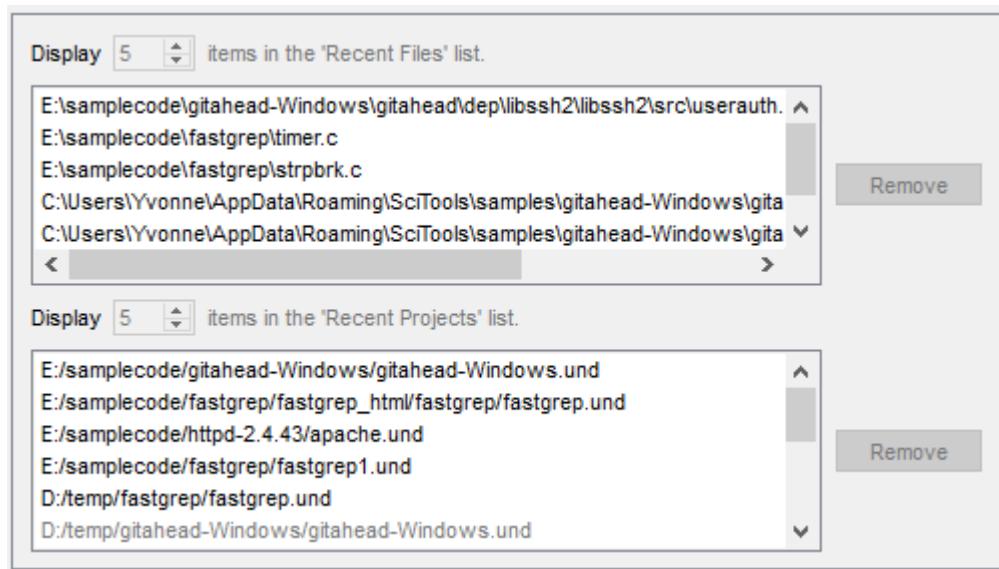


- **Animate Windows/Drawers:** If checked (on by default), opening and closing windows and tabbed areas (drawers) is animated. You can choose a faster or slower **Speed** than the default.
- **Show tabs:** If checked (on by default), tabs are shown at the top of the document area for each of the windows 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.

- **Title Formats:** Choose whether you want filenames in the title areas of windows, tabs, and selector files to be short names, long (full path) names, or relative to the project.
- **Command Window Font:** Controls the font used in the “Run a Command” dialog to display output from the commands you issue.
- **Use the New Project Wizard when creating new projects:** The check in this box causes the New Project Wizard (page 34) to be used when you choose **File > New > Project**. If you uncheck this box, you can specify a project location and filename and then use the full Project Configuration dialog.

## User Interface > Lists Category

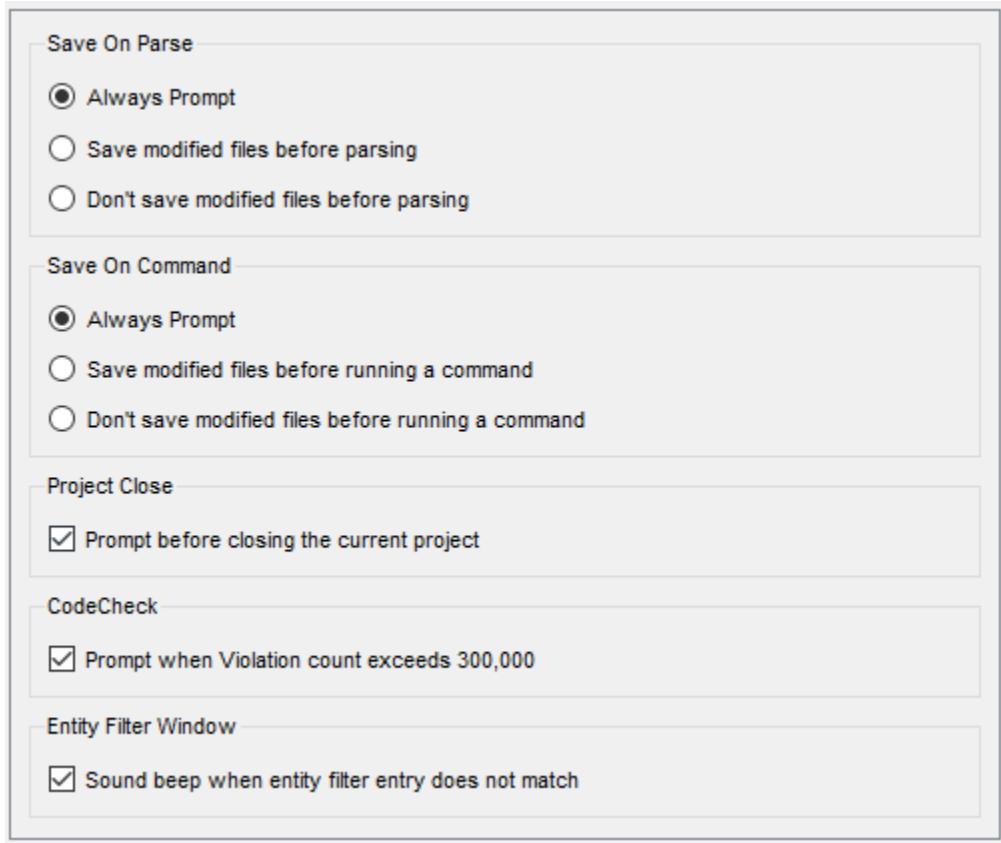
The following options can be set from the **User Interface > Lists** category of the **Tools > Options** dialog:



- **Recent files list:** The default is to show five items in a list of recently used files. You can change that default here. You can remove items from the list that you do not want displayed. Note that you can choose **File > Recent Files > Clear Menu** to clear the history of recent files.
- **Recent projects list:** The default is to show five items in a list of recently used projects. You can change that default here. You can remove items from the list that you do not want displayed. Note that you can choose **File > Recent Projects > Clear Menu** to clear the history of recent projects.

**User Interface > Alerts Category**

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

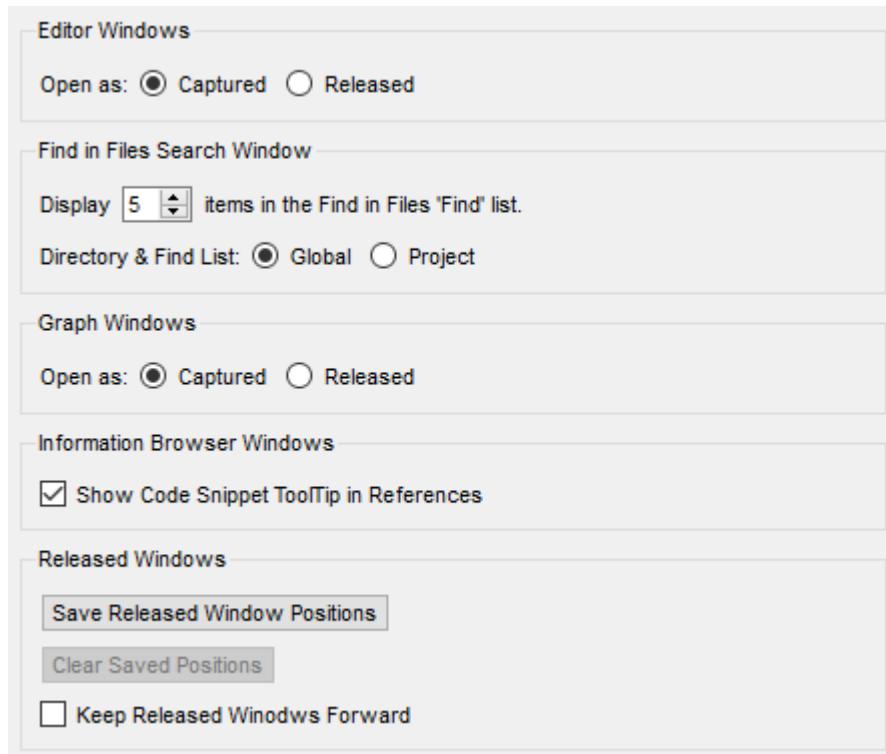


These options can be used to re-enable warnings that you have disabled in a warning dialog box.

- **Save on Parse**: Choose what you want done with changed but unsaved source files when the project 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 a CodeCheck when 300,000 violations are 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 various options for different types of windows.

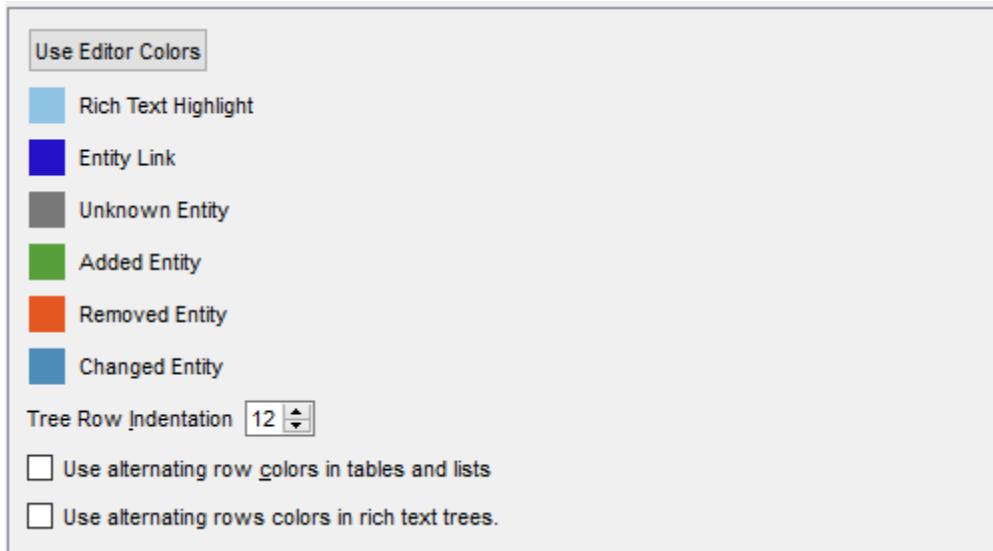
- **Editor Windows:** Choose to open source files as captured windows within the document area of the *Understand* window (MDI windows) or as released windows you can move anywhere on your desktop (SDI windows). The default is captured.
- **Find in Files Search Window:**
  - **Display:** Choose how many items to list in the drop-down list of recent searches. The default is 5.
  - **Directory & Find List:** Choose whether lists of recently used search strings should show all searches or only those searches used with the current project.
- **Graph Windows:** Choose to open graphical views as captured windows within the document area of the *Understand* window (MDI windows) or as released windows you can move anywhere on your desktop (SDI windows). The default is captured.
- **Information Browser Windows: Show Code Snippet ToolTip in References:** Choose whether you want several lines of code to be shown in the hover text when you point to a line number shown in the References list in the Information Browser. If the Information Browser does not show line numbers, click the drop-down arrow next to "References" and choose **Reference > Full**.

- **Released Windows:** Click the **Save Released Window Positions** button if you have released windows from the document area and you want *Understand* to remember the window positions. If you have used this button to save locations, you can use the **Clear Saved Positions** button to forget the locations.
- **Keep Released Windows Forward:** Check this box if you want released windows to stay on top of other windows.

---

**User Interface > Application Styles Category**

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

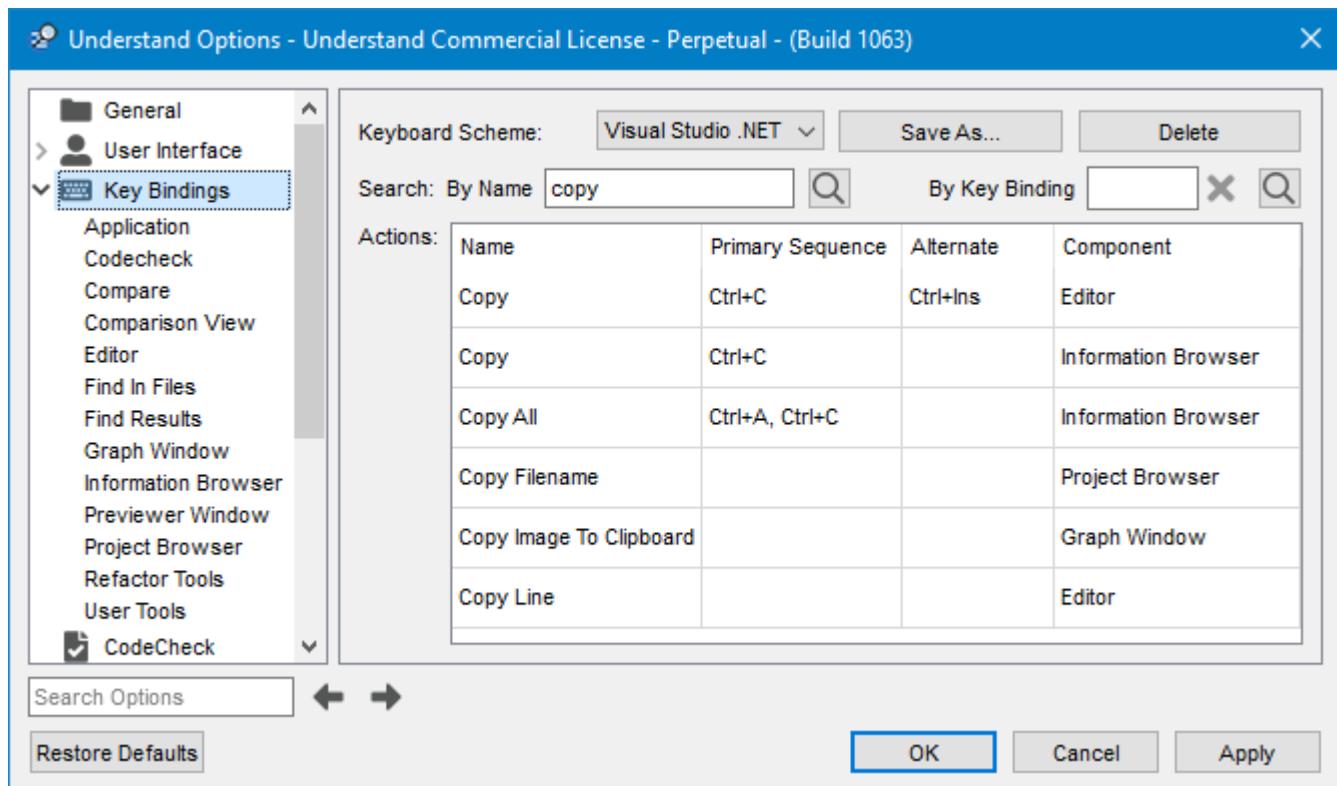


You can choose various options for different types of windows.

- **Use Editor Colors:** Click this button to apply the source editor colors from the Editor > Styles category (page 111) to item views, such as the Information Browser.
- **Entity colors:** These colors are used in item views, such as the Information Browser. Click a color square next to an item in the list. Use the Select Color dialog to choose a new color for that item. Other colors come from system settings, especially on Windows or are set in other categories in this 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, such as the results of a CodeCheck, have shading for alternate rows.
- **Use alternating row colors in rich text trees:** By default, formatted results all have the same background color. You can enable a slightly darker background for every second row by checking 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*:

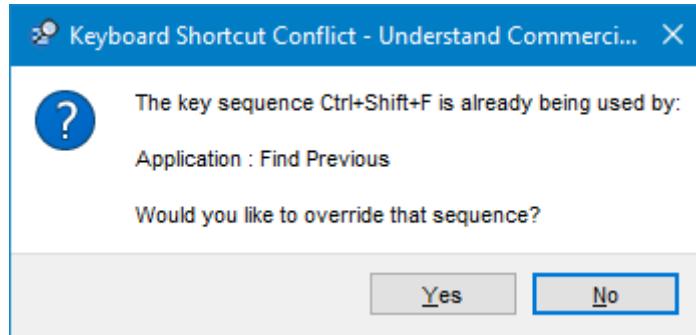


- Keyboard Scheme:** This field allows you to choose groups of keyboard settings that are similar to other applications. The default settings are those native to *Understand*. Other choices are Visual Studio .NET key bindings and the Emacs editor key bindings. If you choose a scheme and click **OK**, 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:

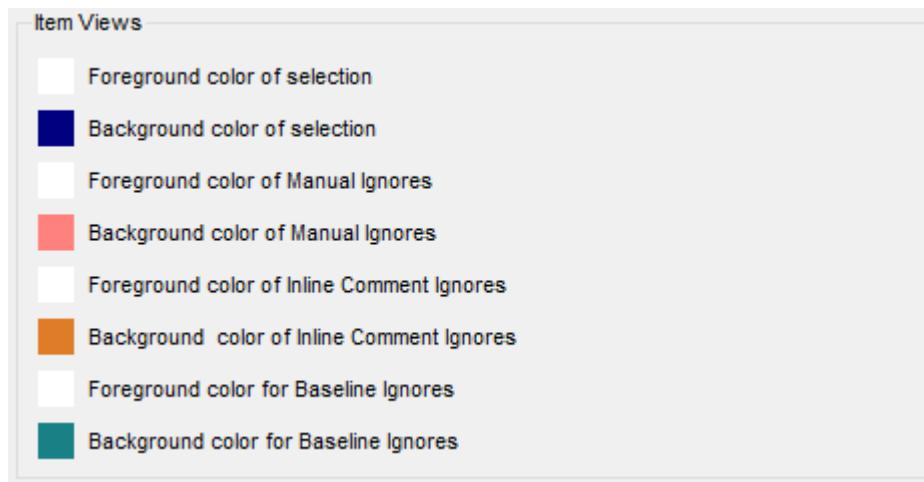
- 1 Use the Component categories or the Search fields to find a command whose key binding you want to change.
- 2 Put your cursor in the **Primary Sequence** or **Alternate** column for the command you want to modify.
- 3 Press the key combination you want to use to perform that action.
- 4 You can't 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**.
- 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.

## CodeCheck Category

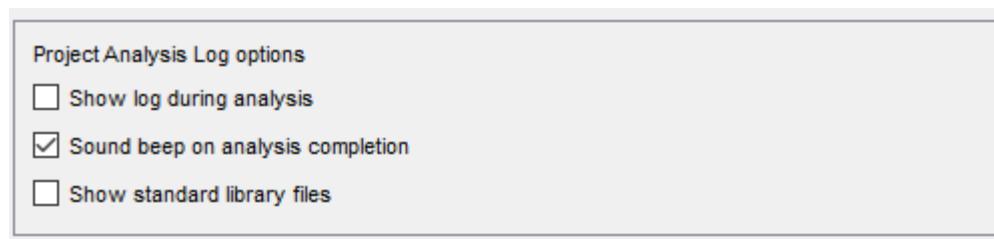
Set colors in the **CodeCheck** category of the **Tools > Options** dialog to specify colors in the CodeCheck window. See *About CodeCheck* on page 241 for more information.



Click a color square next to an item in the list. Use the Select Color dialog to choose a new color for that item.

## Analyze Category

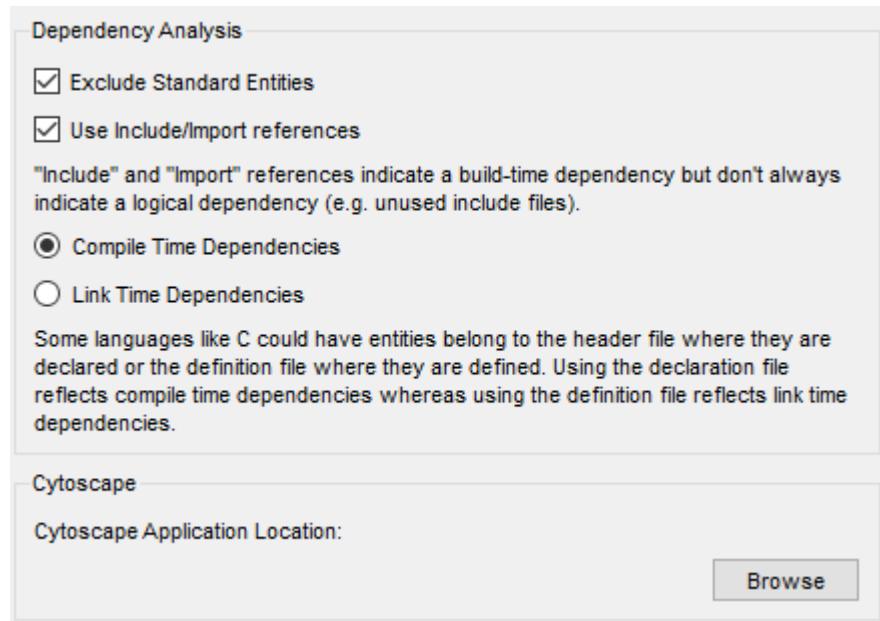
The **Analyze** category of the **Tools > Options** dialog allows you to specify options for how the project is analyzed.



- **Show log during analysis:** By default, the Analysis Log is not shown while the analysis is being performed. If you check this box, the Analysis Log area is shown while an analysis is running.
- **Sound beep on analysis completion:** By default, a beep notifies you when the analysis is complete.
- **Show standard library files:** For languages whose standard libraries are analyzed by *Understand* (such as Ada), if you check this box the standard library files are shown in the Analysis Log. By default, this box is not checked, and the Analysis Log is shorter.

## Dependency Category

The Dependency category of the **Tools > Options** dialog lets you set options related to the Dependency Browser (page 135), dependency graphs (page 201), and dependency exports.



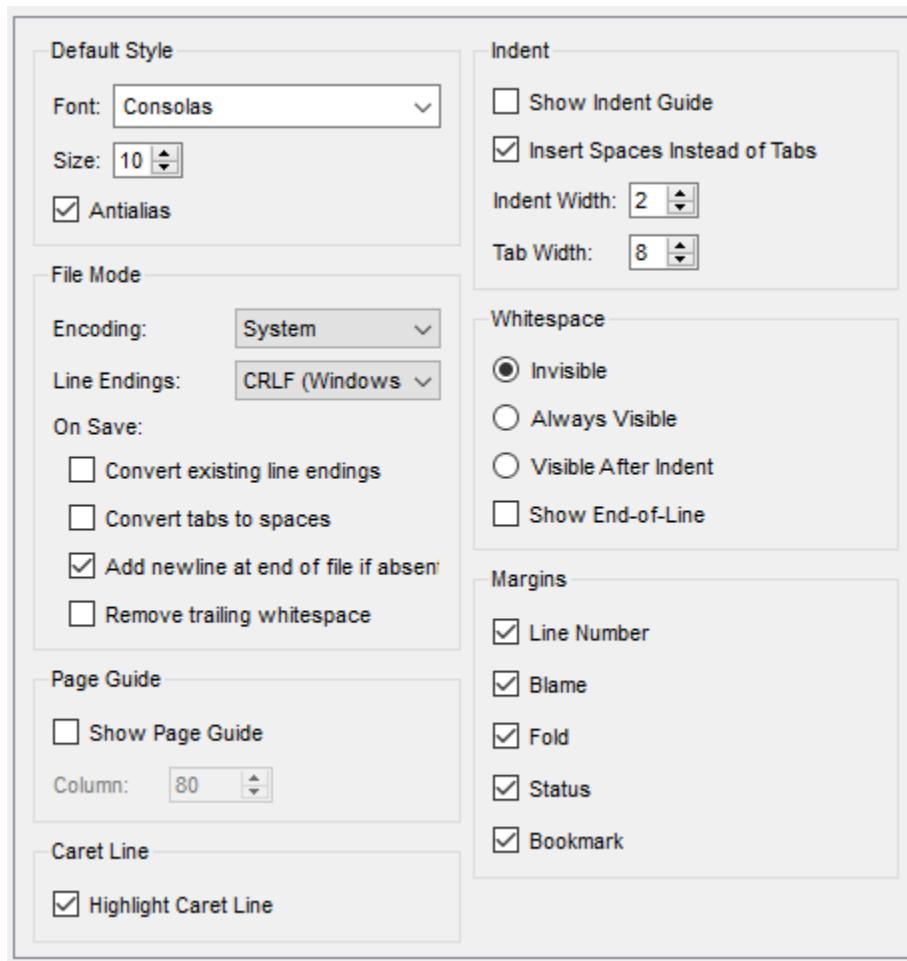
- Exclude Standard Entities:** By default, entities in the standard libraries are excluded from dependency graphs and the dependency browser. Uncheck this option to include such standard entities.
- Use Include/Import References:** By default, “includes” and “imports” are treated as dependencies. However, you may want to omit such relationships from dependency lists if they are required for building but are not logically dependent.
- Compile Time vs. Link Time Dependencies:** Choose whether you want entities to be shown as dependent on the header file in which they are declared (compile time dependency) or the source code file in which they are defined (link time dependency). This applies only to languages, such as C, that make a distinction between declarations and definitions. See *What are Dependencies?* on page 137 for more information.
- Cytoscape Application Location:** You can browse for the location where you installed Cytoscape ([www.cytoscape.org](http://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 141.

## Editor Category

The **Editor** category has general options and options in the following subcategories:

- Editor > Advanced Category on page 107
- Editor > Macros Category on page 110
- Editor > Styles Category on page 111
- Editor > Navigation Category on page 112
- Editor > External Editor Category on page 113

The following options control the general behavior of Source Editor windows. They can be set in the **Editor** category of the **Tools > Options** dialog:



- **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.

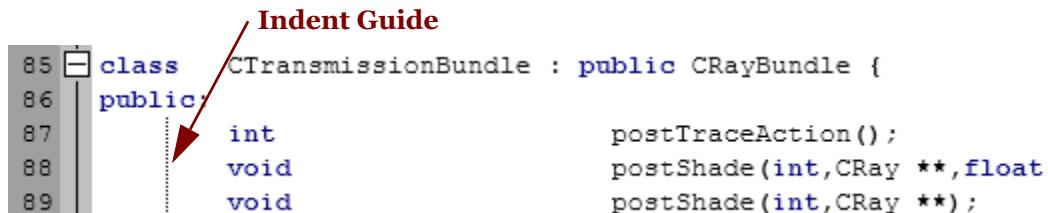
- **File Mode:** Select the type of **Encoding** to use when saving source files and the **Line Endings** character you want used. Many encoding formats are supported. The “System” encoding option uses the same encoding format defined for your operating system. You should change these settings only if 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. When opening a file, a CR, CR, LF sequence is interpreted as a single line ending.
- **Unix** line-endings are terminated with a newline (\n), also called a linefeed (LF).
- **Classic Macintosh** line-endings are terminated with one 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.

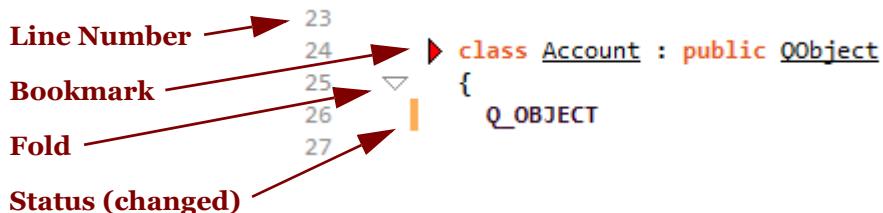
- **Page Guide:** Check the **Show Page Guide** box to display a line similar to the Indent Guide at a defined line width (that is, at the right edge of the code). Set the **Column** to the character width you want to see indicated.
- **Caret Line:** Check the **Highlight Caret Line** box if you want the full line on which your cursor is located to be highlighted.
- **Indent:** Check the **Show Indent Guide** box if you want a dotted line to show the column to which lines should be indented.



By default, the **Insert Spaces Instead of Tabs** box is on, and spaces are added 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. You can set a tab width for a specific file to override the project-wide tab width (see page 182). Also, see *Editor > Advanced Category* on page 107 for advanced indentation options.

- **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:** Show or hide the following left margin columns in Source Editor windows:
  - **Line Number:** (on by default) turns on line numbering in the source view.
  - **Blame:** (off by default) shows Git blame information. See *Exploring Git History* on page 267 for details.
  - **Fold:** (on by default) turns on the ability to “fold” source code entity blocks out of the way.



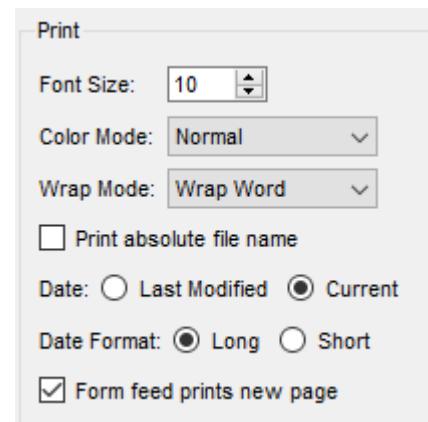
- **Status:** (on by default) shows change bars if the line has been modified but not saved.
- **Bookmark:** (on by default) shows bookmarks (red arrows) next to line numbers.

## Editor > Advanced Category

The following options control more advanced behavior of Source Editor windows. They can be set in the **Editor > Advanced** category of the **Tools > Options** dialog.

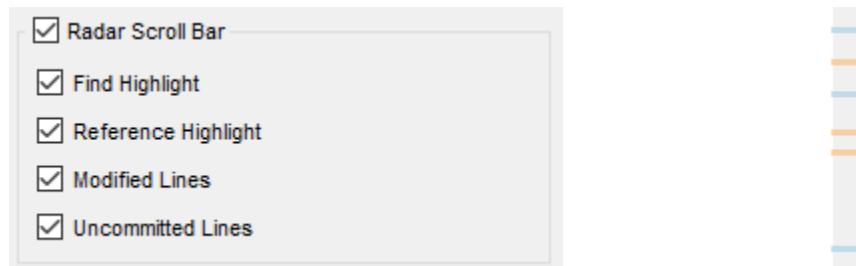
**Print:** The following options control how source code looks when you print it from an editor window:

- **Font Size:** Choose the size of the source code to use for printing. To zoom in and out in an individual source code window, see page 180.
- **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.
  - “Color on White” prints colored code on a white background regardless of the current display appearance.



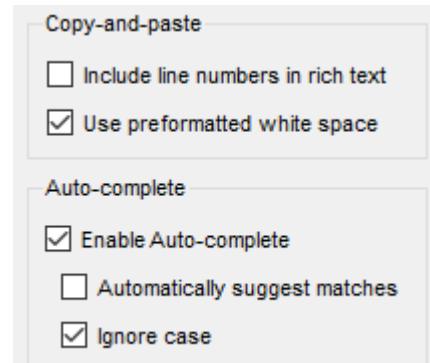
- **Wrap Mode:** Choose the wrap mode you want to use for printing. The default is to wrap words to the next line, but you can choose to truncate lines or wrap at the character level, which breaks words across lines. The line breaks displayed are for printing only; no actual line breaks are added to your source file. See *Line Wrapping* on page 182 to change the wrap mode for screen display.
- **Print absolute file name:** Check this box if you would like the full file path printed at the top of a source file printout, rather than just the filename.
- **Date:** Choose whether to show the date a file was last modified or the current date when printing. The default is the current date.
- **Date Format:** Choose whether to print the date in long or short format. Your system's preferred long and short date format are used.
- **Form feed prints new page:** If this box is checked, a form feed character in the source code file causes a page break. If you uncheck this box, form feed characters are printed as "FF" and no page break occurs.

The **Radar Scroll Bar** area lets you show or hide markers in the scroll bar for a Source Editor window that indicate the location of various content. You can disable all radar highlighting by unchecking the **Radar Scroll Bar** box. The **Find Highlight** box controls highlighting for strings found with Ctrl+F (blue by default). The **Reference Highlight** box controls highlighting for entities found by reference, for example, using the Information Browser (blue by default). The **Modified Lines** box controls highlighting of lines that have been modified but not saved (orange by default). The **Uncommitted Lines** controls highlighting of lines that have been saved but not committed in Git (see *Exploring Git History* on page 267 for details.). The colors of these highlights can be changed using the *Editor > Styles Category* on page 111.

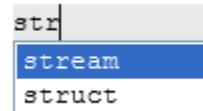


The **Copy-and-paste** area lets you control how text is formatted when you copy and paste code into a word processor.

- **Include line numbers in rich text** pastes line numbers (in bold). HTML is used to format the pasted text. This option is off by default.
- **Use preformatted white space** pastes code using HTML <pre> tags to preserve whitespace. If you disable this option, whitespace is preserved using &nbsp; (non-breaking space) and <br> tags. Some applications may not respect the <pre> tag, in which case you can disable this option to force the formatting to match.

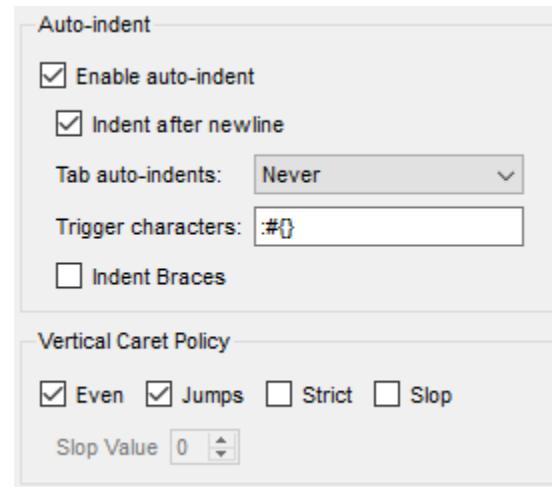


The **Auto-complete** options provide for auto-completion of keyword and entities you type in the editor. As you type, words are shown below your text. You can arrow down through the list and press Enter to choose a suggestion.



- **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.



- **Indent after newline:** If this box is checked, when you start a new line, an indent is added so that you begin typing directly below the first character in the previous line. If unchecked, the cursor is always in the first column on new lines.
- **Tab auto-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 braces:** If you check this box, the automatic indenting formats code with braces as in the following example:

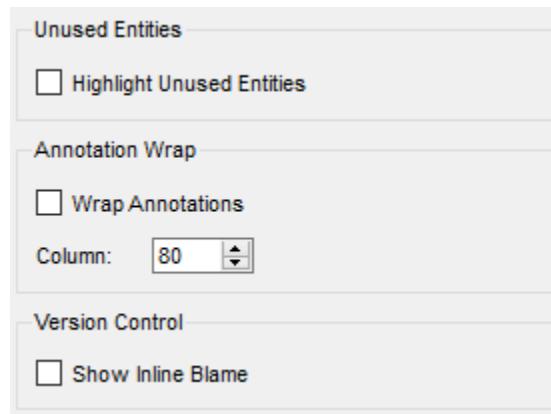
```
if (true)
{
    // block of code here
}
```

The **Vertical Caret Policy** fields control how the Source Editor scrolls as the text cursor or current location highlight moves up and down. Use these fields to optimize the amount of context you see when the Source Editor jumps to a new location. Most users do not need to modify these settings. For details, see the descriptions of interactions between these fields at [www.scintilla.org/ScintillaDoc.html#SCI\\_SETYCARETPOLICY](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 use a colored background to highlight entities that are never used. 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 code in that function has a gray background if you enable this feature.

The **Annotation Wrap** fields let you cause annotation text to be wrapped at the specified **Column**. This feature is off by default.



The **Show Inline Blame** field in the **Version Control** area shows the blame information from Git for each line. See *Exploring Git History* on page 267 for details.

---

## Editor > Macros Category

You can record, save, and replay Source Editor macros as described page 183. After you have saved Source Editor macros, you can rename and delete macros using the Options dialog. Follow these steps:

- 1 Choose **Tools > Options**, expand the **Editor** category, and select the **Macros** category.
- 2 In the top box, choose the macro you want to configure.
- 3 Click **Edit** if you want to rename the macro or assign a different key sequence to trigger it. Note that you cannot edit the actions performed by the macro. To modify the actions, record a new macro.
- 4 Click **Remove** if you want to delete the macro.

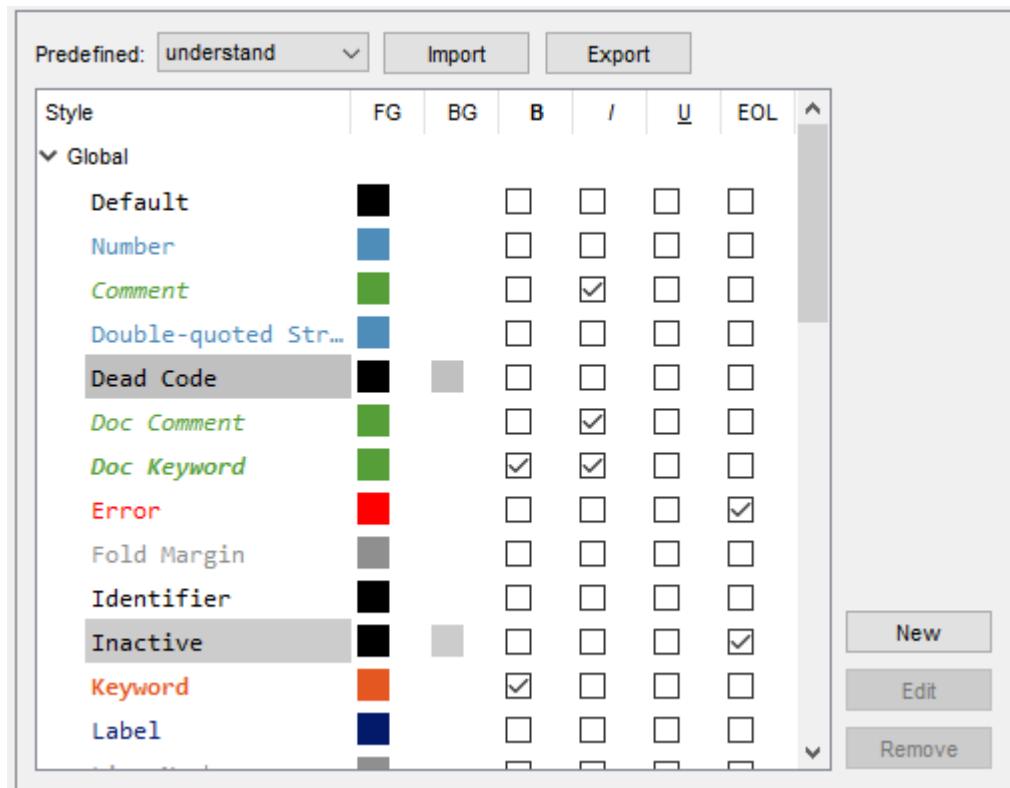
## Editor > Styles Category

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.

*Understand* also switches between dark mode and light mode based on your system settings. These color themes change based on which mode you are in.

To choose a color scheme with a set of defined colors, choose a scheme from the **Predefined** list. The default scheme is “understand”.

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.

You can use the **Import** and **Export** buttons to save your Editor style settings to an *Understand Theme (\*.lua)* file. This allows you to share styles between computers.

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

- **Dark orange text:** Used for language and preprocessor keywords
- **Blue text:** Used for characters and character strings
- **Italic green text:** Used for comments
- **Black text:** Used for all other source text
- **White background:** Used for most source text
- **Gray background:** Used for inactive lines of code
- **Blue background:** Used to highlight text in Search

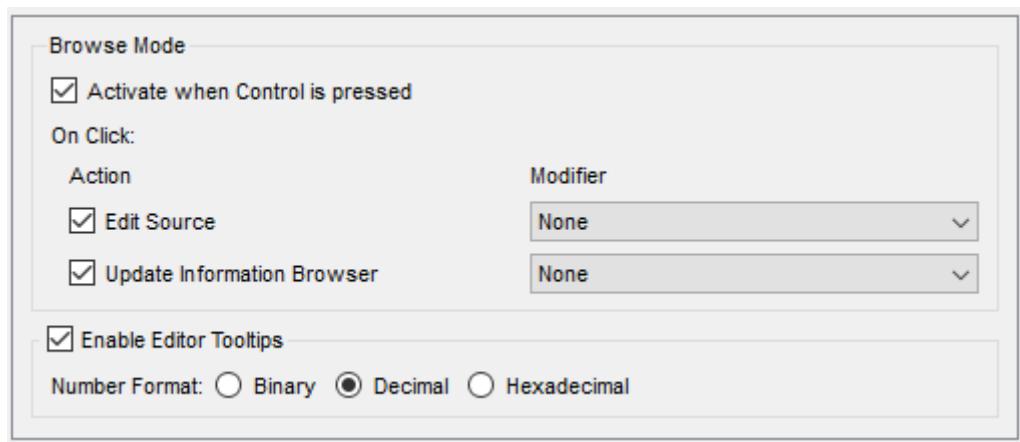
Additional items are available for customization depending on your source code language. For example, with C++, you can customize class, enumerator, namespace, and many other categories of names. With Pascal, you can customize the colors of module, routine, and type names. With Fortran, you can customize the colors of block, module, subprogram, and type names. With Ada, you can customize the colors of global, local, package, subprogram, 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 169). To open this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Navigation** category.



- **Activate when Control is pressed:** If this is checked (on by default), Source Editor windows use Browse Mode if you hold 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 the entity. 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.
- **Number Format:** Choose whether to display numeric values as decimal, binary, or hexadecimal values in hover text for source code. For example, variables initialized with a numeric literal and enumerated values would show such hover text.

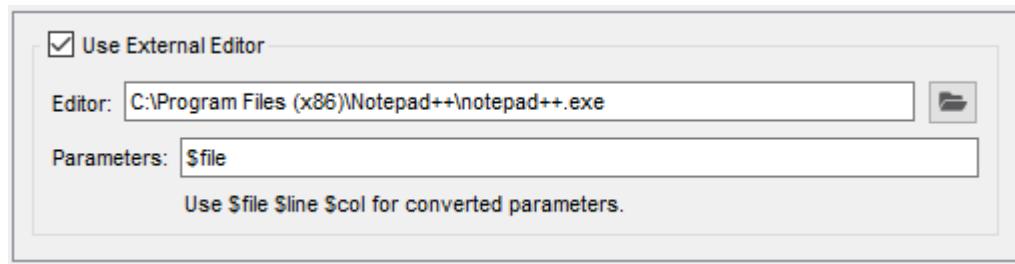
## Editor > External Editor Category

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 **Tools > Options**. Expand the **Editor** category, and select the **External Editor** category.
- 2 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
```

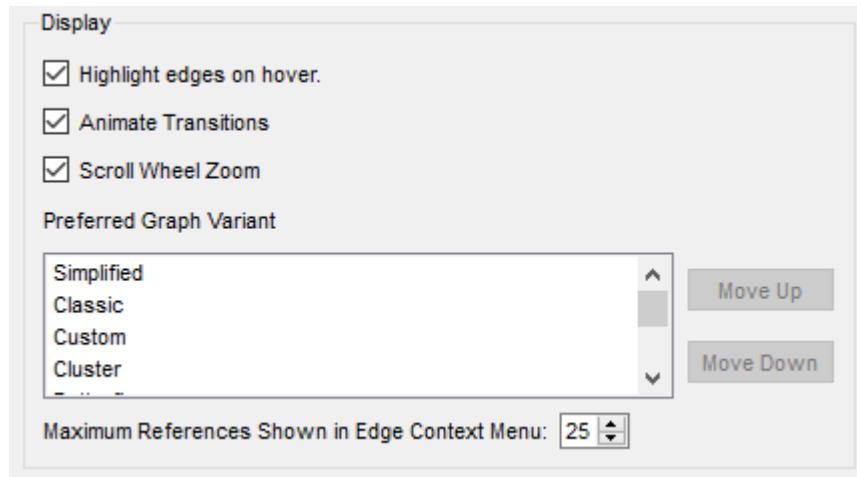
For the TextPad editor on Windows, the **Editor** is most likely c:\Program Files\textpad4\textpad.exe, and the **Parameters** should be as follows:

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

The *Understand* context menus (also called right-click menus) can be made usable in external editors. Steps are provided in the SciTools support website. Search the [SciTools Support website](#) for steps to integrate editors such as EMACS, vi, Visual Studio, and SlickEdit.

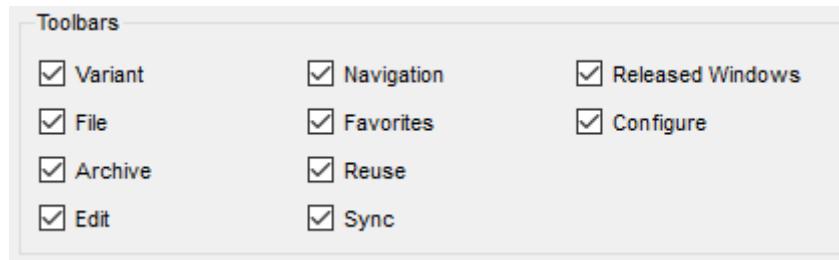
## Graphs Category

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

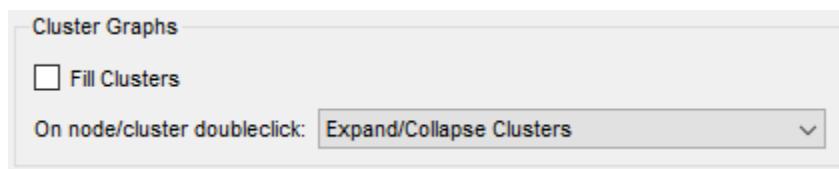


- **Highlight edges on hover:** If this option is enabled, relationships (connecting lines) within a graph are highlighted when your mouse cursor hovers over the relationship. This makes it easier to distinguish between overlapping relationships. Text describing the relationship is always shown when your mouse cursor hovers over a relationship; this option does not affect display of the relationship description. See page 201.
- **Animate Transitions:** By default, when you make a change to a graph (such as expanding or compressing a node or changing settings in the Graph Customizer area), the transition to reorganize the graph and display children of the expanded node is animated. Uncheck this box if you want to omit the animated transition.
- **Scroll Wheel Zoom:** By default the mouse scroll wheel can be used to zoom in and out. Uncheck this box to cause the mouse scroll wheel to scroll up and down within a graph.
- **Preferred Graph Variant:** By default, the Simplified variant (if available) is opened when you select a graph category. This list shows the order in which *Understand* looks for a graph variant to open. You can select a variant and click **Move Up** or **Move Down** to change the preference. See page 220 for more about graph variants.
- **Maximum References Shown in Edge Context Menu:** If you right-click on any line between nodes in a cluster graph, a list of the relationships represented by this edge is provided. By default, up to 25 relationships per node are shown. You can make this limit higher or lower. The maximum allowed value is 99.

The **Toolbars** area of this category lets you select which parts of the toolbar to display for graphs. See page 222 for the toolbar icons controlled by these check boxes



The **Cluster Graphs** area lets you customize the behavior of cluster graphs.



- **Fill Clusters:** By default, the background of a cluster is transparent. Check this box to add a colored background for clusters.
- **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.

The **Cluster Graph Styles** area of this category lets you customize the display colors, shapes, and arrows for all customizable graphs, which include architecture dependency, cluster, comparison, and control flow graphs.

**Cluster Graph Styles**

All matching styles are applied, with styles at the top taking priority. Styles inherit settings from the "Default" style and only overwrite non-default values. For edges, custom reference string style colors stack, creating a composite of all matching custom styles.

**Nodes**

Style	Fill	Line	Shape	
Identical (Compare)	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span> box	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Added (Compare)	<span style="background-color: #2e8b57; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #2e8b57; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #2e8b57; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span> box	<span style="background-color: #2e8b57; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Removed (Comp...)	<span style="background-color: #c8512e; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #c8512e; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #c8512e; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span> box	<span style="background-color: #c8512e; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Unreachable (Co...)	<span style="background-color: #c00000; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #c00000; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #c00000; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span> box	<span style="background-color: #c00000; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Start (Control Flo...)	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span> box	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>

**Edges**

Style	Line	Arrow Head	ArrowTail	Line Style	
Identical (Compare)	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="color: #4f81bd;">▶</span> normal	none	solid	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Added (Compare)	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="color: #4f81bd;">▶</span> normal	none	solid	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Removed (Comp...)	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="color: #4f81bd;">▶</span> normal	none	dotted	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
Yes (Control Flow)	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="color: #4f81bd;">▶</span> normal	none	solid	<span style="background-color: #4f81bd; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>
No (Control Flow)	<span style="background-color: #c00000; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>	<span style="color: #c00000;">▶</span> normal	none	solid	<span style="background-color: #c00000; border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span>

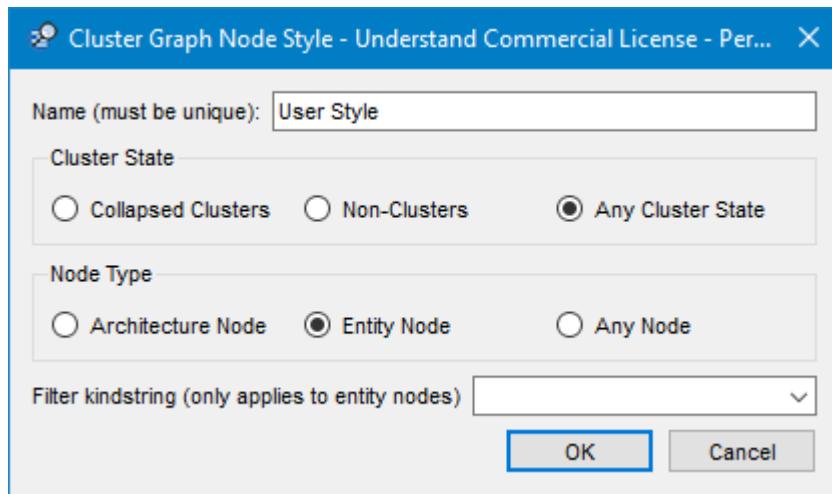
To change the color of a type of node or edge (line between entities), click the box in the **Fill** column or the **Line** column. To change the shape of the box for an entity or architecture node, use the drop-down list to select a different **Shape**. To change the ends of an arrow, select a different **Arrow Head** or **Arrow Tail**.

You can use the **Move Up** and **Move Down** buttons to change the order of the styles.

You can create custom styles for nodes as follows:

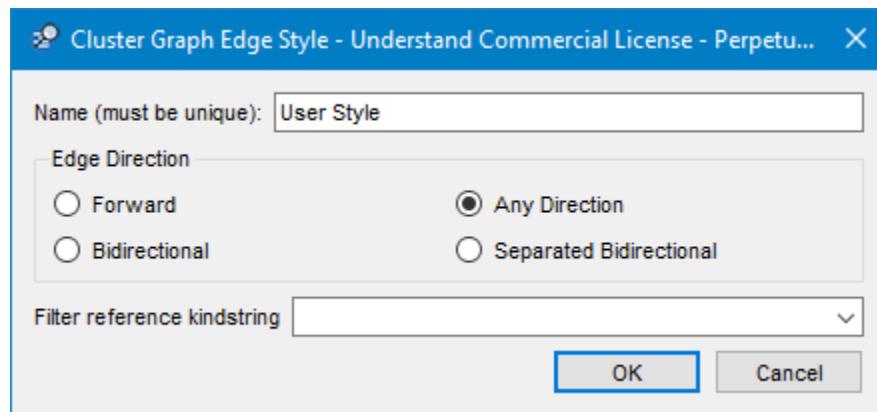
- 1 Click the **New** button for Nodes.
- 2 Type a unique name for the new style.
- 3 Choose whether this style should apply to only collapsed clusters, only non-clusters, or any cluster.
- 4 Choose whether this style should apply to only architecture nodes, only entity nodes, or all nodes.
- 5 If this is an entity node style, you can choose a way to filter entities that should have this style. Several sample filters are provided, and you can modify the suggested filters to create your own. For example, the "local object" filter applies the style to

locally-defined objects only. The “type ~unnamed” filter applies the style to entities whose type is not unnamed.



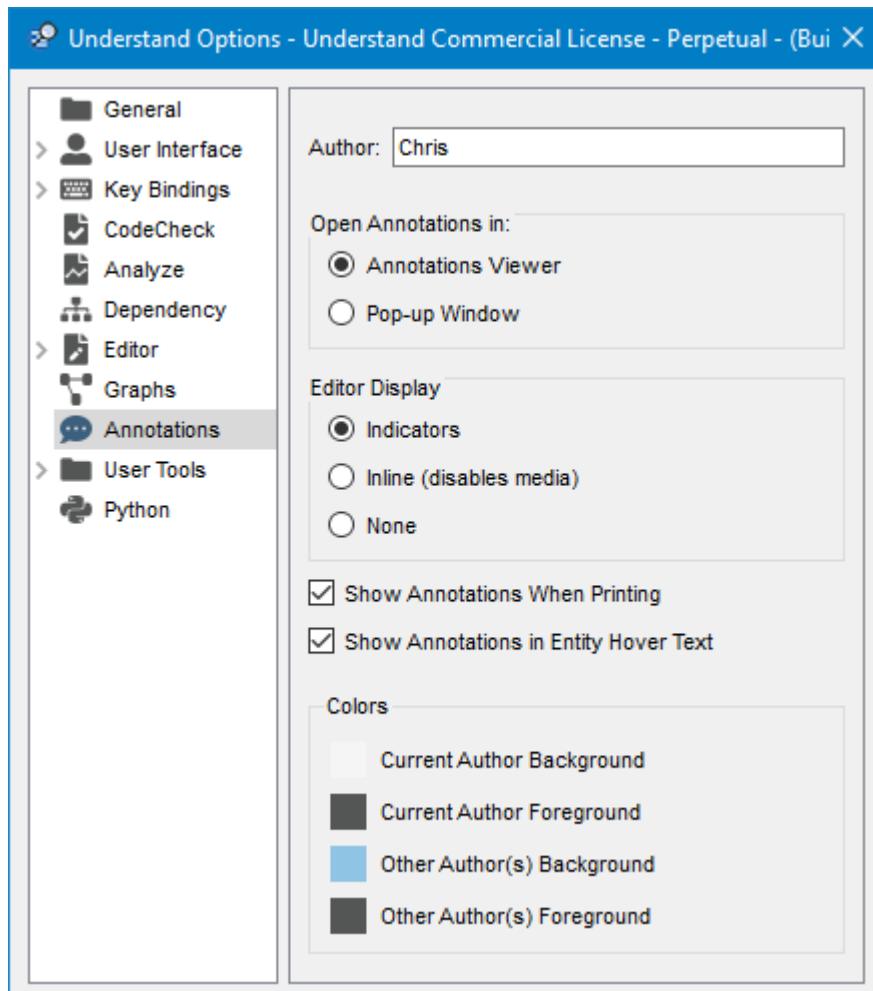
You can create custom styles for edges as follows:

- 1 Click the **New** button for Edges.
- 2 Type a unique name for the new style.
- 3 Choose whether this style should apply to only forward references, bi-directional references, separated bi-directional references, or all references.
- 4 You can choose a way to filter the edges that should have this style. Several sample filters are provided, and you can modify the suggested filters to create your own. For example, the “inherit, inheritby” filter applies the style only to inheritance relationships. The “call ~inactive, callby ~” filter applies the style only to call and callby relationships that are active.



## Annotations Category

The **Annotations** category of the **Tools > Options** dialog lets you control how annotations are displayed and edited. See page 186 for details on using annotations.



- **Author:** Type your name or the username you want to be associated with the annotations you create.
- **Open Annotations in:** By default, when you point to an annotation indicator, the annotation is shown in a pop-up area (unless the Annotations Viewer is open) and when you click on an annotation indicator or in the Annotations Browser, the annotation is shown in the **Annotations Viewer**. If you select **Pop-Up Window** instead, the Annotations Viewer does not open when you click on an annotation indicator; only the pop-up area is shown.
- **Editor Display:** By default, an annotation Indicator (three dots in a speech bubble icon) is shown in the right margin of the Source Editor next to any line that has an annotation. If you select **Inline**, the annotation is shown below the annotated line in the source code. If you

select **None**, annotations are not visible. Note that annotations at the file level and annotations that contain an attached file are not visible in Inline mode unless you open the Annotation Viewer.

- **Show Annotations When Printing:** If this box is checked, when you print a source code file, any annotations are printed in Inline mode no matter how they are displayed in Source Editor windows.
- **Show Annotations in Entity Hover Text:** If this box is checked, annotations are shown if you point to any place where the associated entity is used in the code for two seconds.
- **Current Author Background:** Click on the colored block to change the background color in the Annotations Viewer and pop-up annotations for annotations you added.
- **Current Author Foreground:** Click on the colored block to change the text color in the Annotations Viewer and pop-up annotations for annotations you added.
- **Other Author(s) Background:** Click on the colored block to change the background color in the Annotations Viewer and pop-up annotations for annotations other users added.
- **Other Author(s) Foreground:** Click on the colored block to change the text color in the Annotations Viewer and pop-up annotations for annotations other users added.

Annotations are stored in JSON files in the `<project>.und` directory. Any attached files are stored in the media subdirectory of the `<project>.und` directory.

## User Tools Category

See *Configuring Tools* on page 272 for information about how to configure tools and provide commands to run these tools in the right-click menus, Tools menu, and toolbar.

## Python Category

The **Python** category of the **Tools > Options** dialog lets you choose which Python interpreter is used to run user tool scripts that use *Understand's* Python API.

To choose a different Python interpreter, specify the path to the Python shared library. Leave blank to use the bundled Python.

On Windows, this library is typically located in the root of the Python install directory and named python3x.dll.

*note: Understand must be restarted for this change to take effect*

Library Path:

...

By default, the Python interpreter bundled with *Understand* is used. For Windows, this defaults to C:\Program Files\SciTools\bin\pc-win64\Python\Python-3.8\python.exe.

To change the Python interpreter, click ... and browse for your Python executable.

See *Python Options* on page 84 to set the version of Python used when analyzing Python source code.

---

## Chapter 5      Exploring Your Codebase

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 146.

Details on the use and operation of the Source Editor is contained in the chapter *Editing Your Source* on page 165.

This chapter contains the following sections:

Section	Page
PLEASE RIGHT-CLICK	122
Various Windows Explained...	123
Entity Filter	124
Information Browser	126
Project Browser	132
Exploring a Hierarchy	134
Dependency Browser	135
Favorites	142

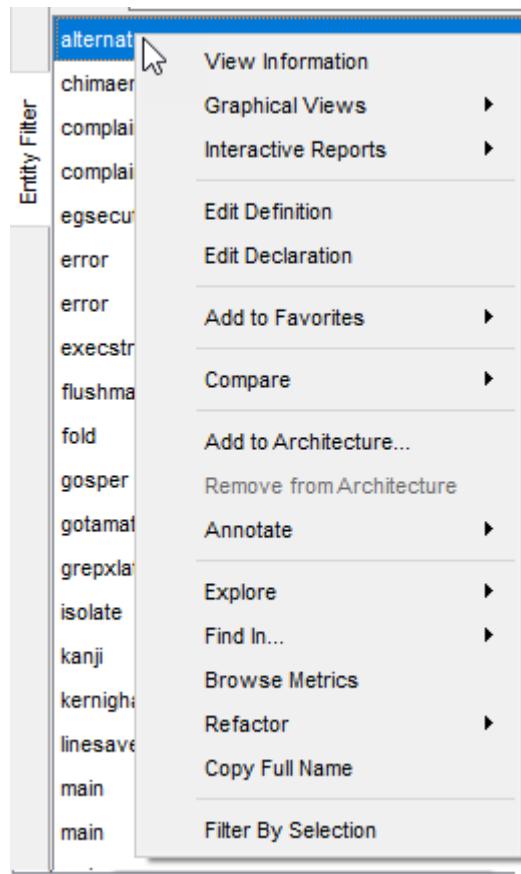
## PLEASE RIGHT-CLICK

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.

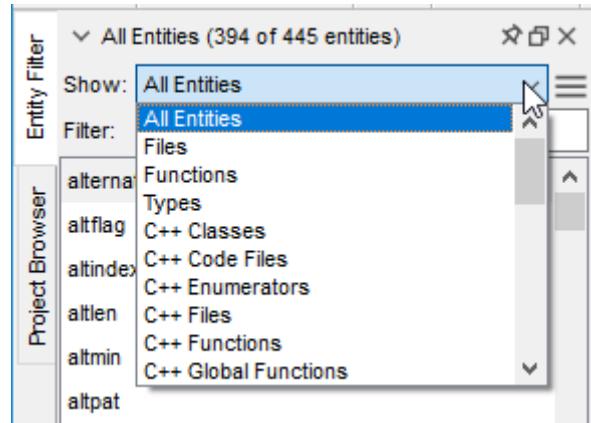
The tools available for finding and exploring entities are:

- **Analysis Log:** Shows results of project analysis, including any errors and warnings. See page 88.
- **Architecture Browser:** Defines named regions and views of the project. See Chapter 8.
- **Bookmarks:** Show and jump to bookmarks. See page 184.
- **Contextual Information Sidebar:** Show context information about the current source editor file. See page 164.
- **Dependency Browser:** Lets you browse dependency relationships. See page 135.
- **Entity Filter:** Provides an alphabetic list of entities of the selected type. See page 124.
- **Entity Locator:** Lets you filter all entities in a project in complex ways. See page 155.
- **Exploring View:** Lets you browse a relationship hierarchy. See page 134.
- **Favorites:** Lets you provide quick links to frequently-used entities. See page 142.
- **Find in Files:** Searches multiple files. See page 150.
- **Information Browser:** Provides an explorer for entity characteristics and connections. See page 126.
- **Previewer:** Provides a quick way to view code in successive locations. See page 179.
- **Project Browser:** Lets you browse a hierarchical file list. See page 132.
- **Source Editor:** Shows source code. See page 165.
- **Scope list:** Lists the functions or similar constructs in a file. See page 167.
- **Graphical Views:** Shows connections and structures of entities. See Chapter 10.
- **Metrics Browser:** Generate statistics about entities and architectures. See page 208.
- **Window Selector:** Jump to a window that is open. See page 161.
- **Annotations View:** Viewing annotations in the project. See page 186.

## Entity Filter

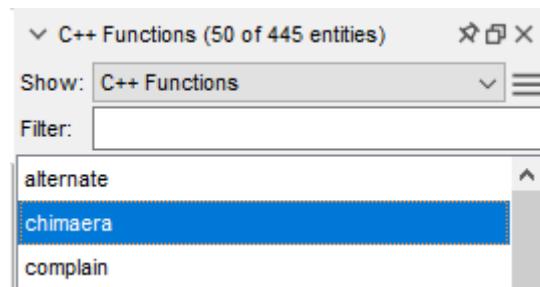
The *Entity Filter* provides a quick list of the selected entity type. You can filter this list to match a text string.

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 hamburger menu 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 hamburger menu 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 hamburger menu  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 157 for examples.
- **Regular Expression:** With this option selected, you can use Unix-style regular expressions. See page 157 for an overview.

To see only unknown or unresolved entities, click the hamburger menu and choose **Filter Unresolved Entities > Hide Resolved Entities**. To see only resolved entities, click the hamburger menu and choose **Filter Unresolved Entities > Hide Unresolved Entities**. To see both resolved and unresolved entities, choose **Filter Unresolved Entities > No Filter**.

When you are finished using a filter and want to see all the entities for the selected type, click the hamburger menu 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 hamburger menu .

You can select from filters you have used by right-clicking the Filter area and choosing from the **Most Recent Filters** list. Filters are shown in this list if you have selected any entity found using a filter.

## 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 hamburger menu 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 hamburger menu and choose **File Name as > Relative Name** or **File Name as > Long Name**.

By default, only the name of a function or method is shown. To also show the parameters of such entities, click the hamburger menu and choose **Show Parameters as > Full Parameters** or **Show Parameters as > Short Parameters**.

## Root Entity Types

There are entity types for most languages in the **Show** drop-down list that contain “Root” in the name, such as **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 other entities, but does not list entities that call other entities. These are low-level routines.
- **Root IncludeBys:** Lists only files that are included by other files, but does not list files that include other files. These are low-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 any other program units. (Ada)

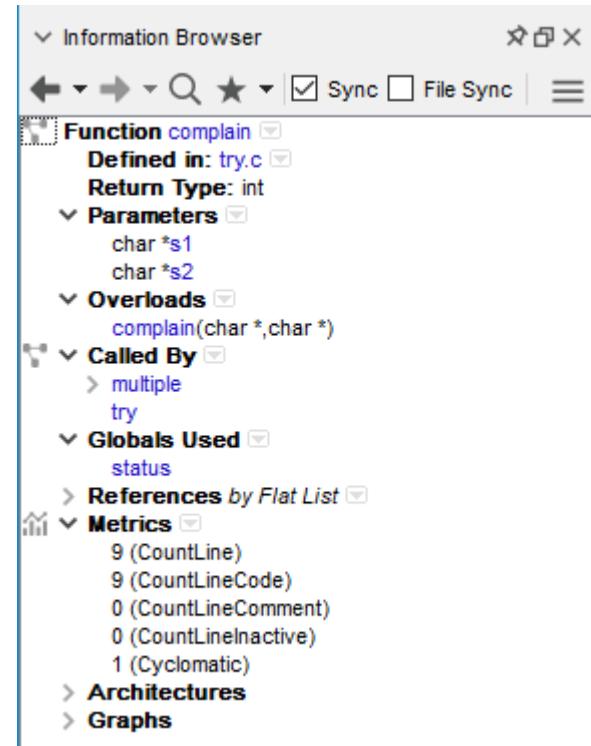
## Information Browser

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.

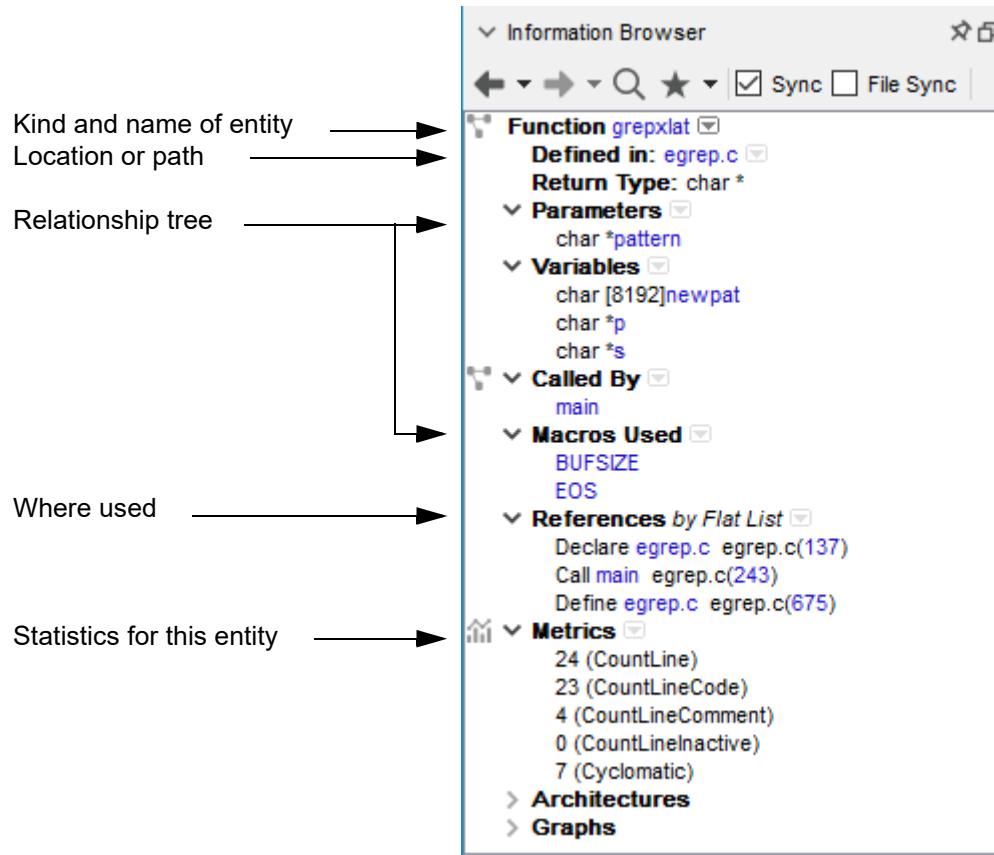
If the *Information Browser* isn’t open, you can right-click on an item anywhere in *Understand* and choose **View Information**. Or, choose **View > Information Browser** from the menus.

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.



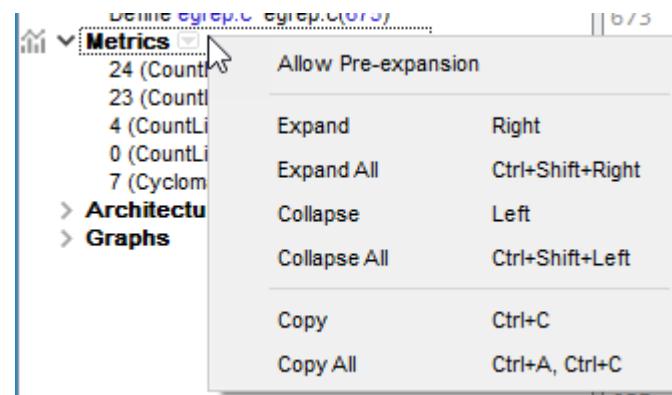
## Drilling Down a Relationship

Drilling down the tree works as expected (mostly). To expand a tree, click on the > arrow. To close the tree click on the down arrow.

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 131 for details on the other options in this context menu.



## Displaying More or Less Information

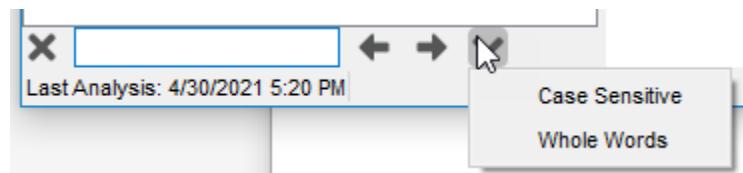
If you click the  icon next to a bold heading such as **Calls, Called By** or **References** in the Information Browser (or right-click on the heading), you'll see options that let you modify how that entity is listed. These options include:

- **Allow Pre-expansion:** If enabled, the Metrics node can stay open when you change entities. See *Viewing Metrics* on page 130.
- **Defn Name:** Controls whether the file in which a called or called by function is defined is not shown or is shown using the short, long, or relative file path.
- **Filename:** Controls whether the reference format is short, long, or relative to the project folder.
- **Fullscreen:** If checked, the fully-qualified name of the entity is shown.
- **Group by:** For C++ classes, choose whether to sort class members by the type of access available (public or private) or the kind of member (function or object).
- **Inactive:** Controls whether inactive references are listed.
- **Macros:** Controls whether the Macros Used list includes macros that behave as functions, macros that behave as objects, or both.
- **Parameter:** Lists the parameters.
- **Reference:** Choose “Full” to include the file and line location of the reference.
- **Return Type:** Lists the return type.
- **Show Linkname:** When linked entities exist in multiple languages, display the alternative name if it is different.
- **Sort:** Controls the sort order of the list.
- **Type:** If checked, the datatype is shown.
- **View by:** For lists of references, you can choose whether to display a flat list of references or to group references by the files that contain them or by one of the defined architectures.

## Searching the Information Browser

If you click the magnifying glass 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.



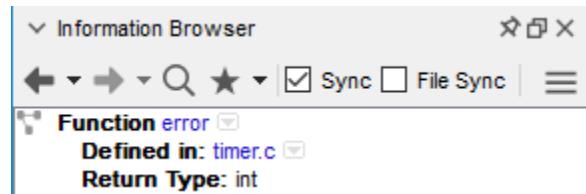
To make the search **Case Sensitive** or to match only **Whole Words**, use the drop-down arrow to select those commands.

## 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.

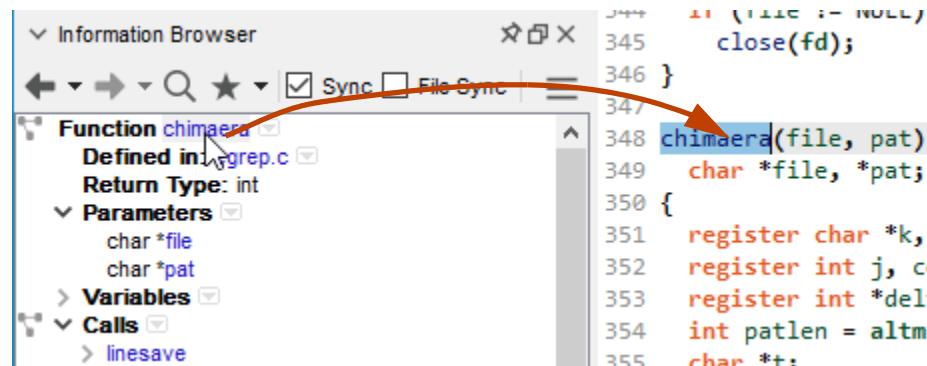
If **Sync** is checked but you want to open a separate Information Browser, hold down the Ctrl key while choosing **View Information** from the right-click menu.

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*.

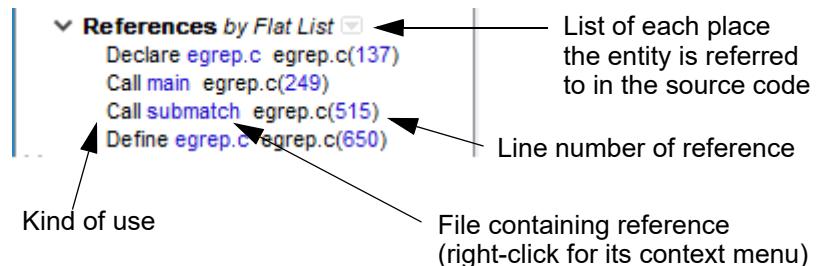


Another way to visit source from any entity you see in *Understand*, is the context menu. Where appropriate, an entity's context menu contains an **Edit Source** (Ctrl+Shift+S) menu item. In some cases, there are separate menus items for **Edit Definition** and **Edit Declaration** (Ctrl+Shift+D) or separate menus for other language-specific locations.

The **Edit Companion File** command opens and cycles through other files with the same name but different extensions. This is useful, for example, with \*.c and \*.h files.

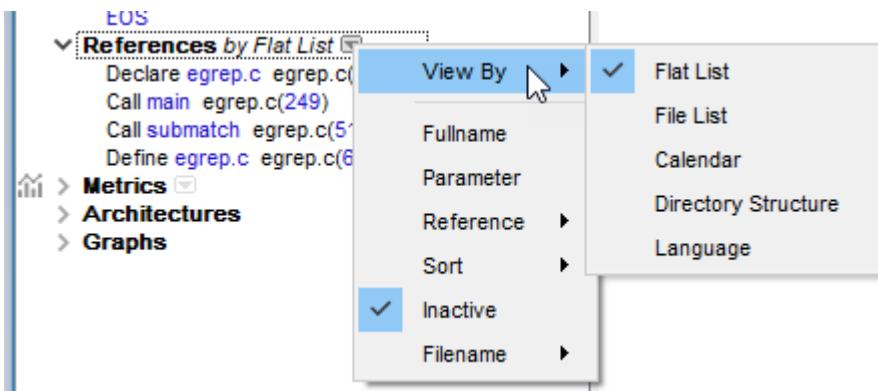
**Visiting References**

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.

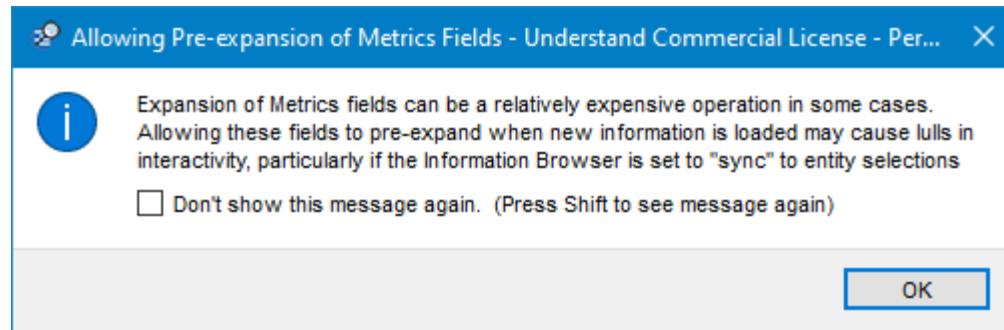
Right-click on the “References” title for the node or click the down-arrow next to the node to choose how to organize the references. Your choices are the default flat list, and all of your architectures.

**Viewing Metrics**

The **Metrics** node on the Information Browser 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 *About Metrics* on page 206 for details on metrics.

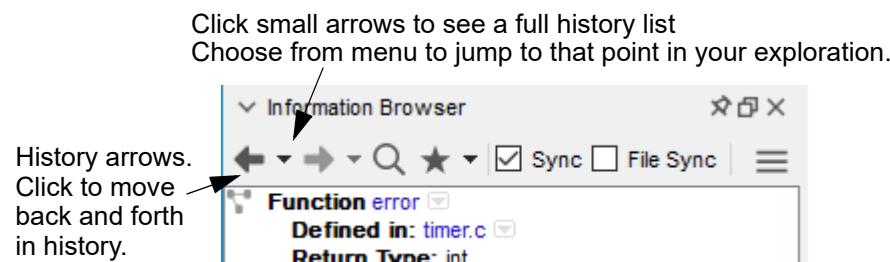
#### Saving and Printing Information Browser Text

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**. For entities with a class path and files, the **Copy** command copies the short name and the **Copy Full Name** command copies the full class path or file path.

#### 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:

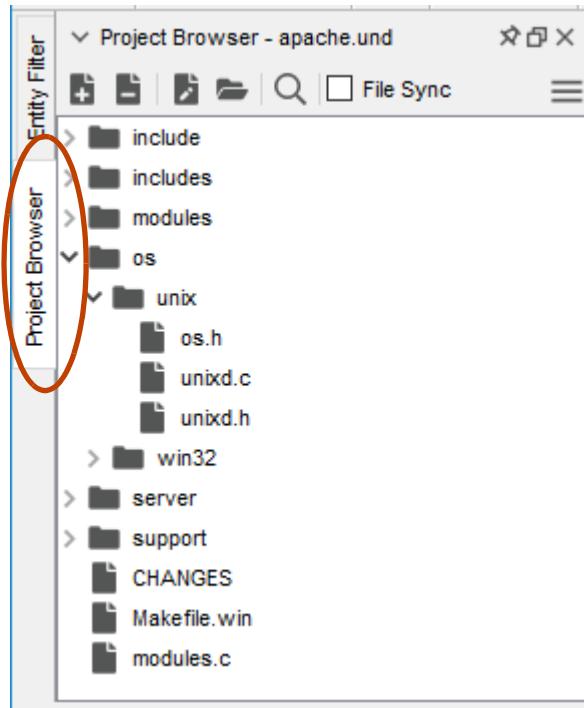


Use the right and left arrows to move back and forward in the history list. The down-arrows show the whole list.

You can choose **Clear History** from the drop-down history list to clear the browsing history.

## 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 when you click on a file in this view offer the same commands as other views such as the Information Browser or Entity Filer.

The context menu when you click the background of this view offers a number of options.

Expand all nodes of project tree	Expand All	Ctrl+Shift+Right
Collapse all nodes of project tree	Collapse All	Ctrl+Shift+Left
Add a file to the project	Add Files...	Ctrl+Shift+F
Remove selected file from project	Remove	Ctrl+Shift+Del
Open file in Source Editor	Edit	Ctrl+Shift+E
Open file with default OS tool	Open Externally	Ctrl+Shift+O
Copy filename to clipboard	Copy Filename	
Change sort order of files	Sort By	▶
Hide certain files	Hide By	▶

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

The **Add Files** 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**.

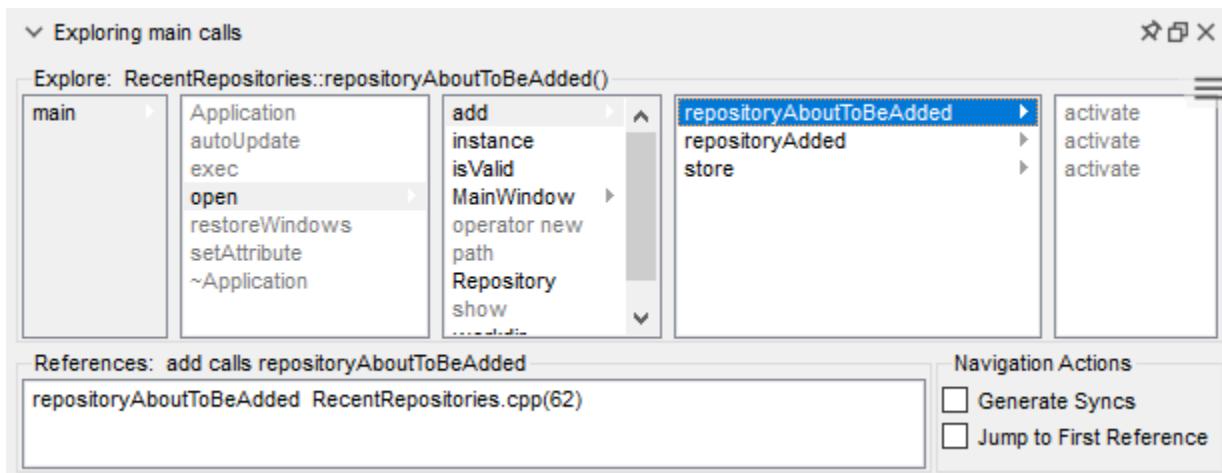
The **Open Externally** command opens an operating system dependent tool for the directory or file. For example, on Windows it opens a directory using the Windows Explorer. For a file it opens the default tool for the file extension.

The **Sort By** command lets you organize the list alphabetically by filename or by file extension.

The **Hide By** command is useful for viewing which files have not yet been added to a custom architecture. It filters out any files that have been added to the selected architecture.

## 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.

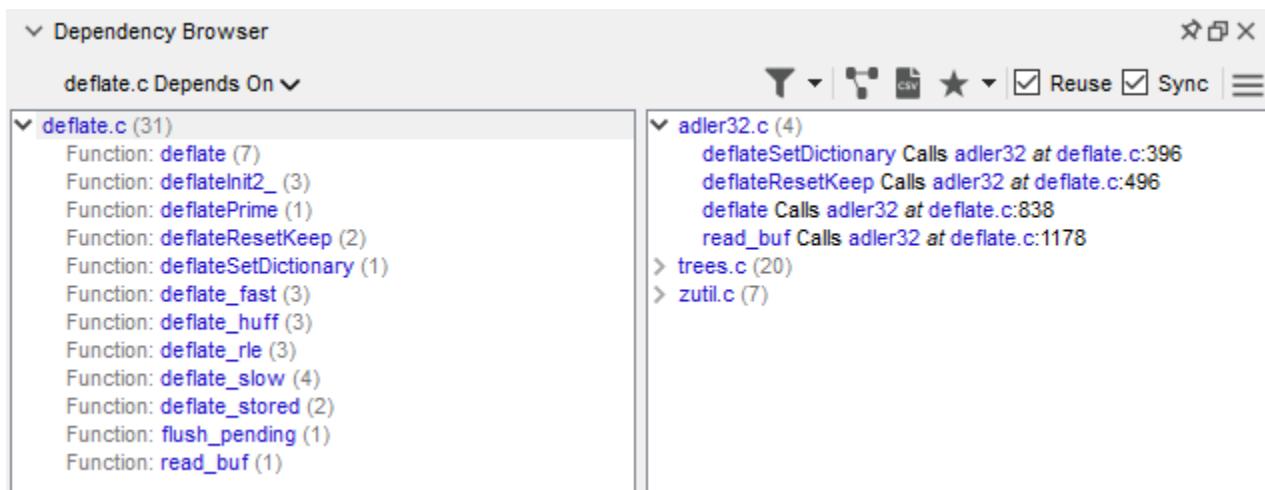
If you check the **Jump to First Reference** box, then the Source Editor automatically displays the initial reference to any entity you select in the Exploring window. Holding the Ctrl key down temporarily activates this behavior.

Click the hamburger menu if you want to enable any of the following display options:

- **Show Long Name:** Shows the long name of each call.
- **Show Parameters:** Shows the parameter list for each call.

## Dependency Browser

The Dependency Browser lets you examine which items are dependent on others. You can use the Dependency Browser with architecture nodes, files, and most entity types.



To open the Dependency Browser, right-click on any architecture node, file, or entity anywhere in *Understand*. For example, select an entity in the Entity Filter, Information Browser, or a graphical view. Choose **View Dependencies** from the context menu. Or, open the Dependency Browser by choosing **View > Dependency Browser**.

The left panel is the Scope View, which shows the item you selected and its children. From the drop-down list above the left panel, choose whether to list items that the selected node or entity **Depends On**, is **Depended On By**, has **References From**, or has **References To**. (Some of these options are not available for certain node or entity types.)

The right panel is the Entity View, which shows items with dependencies on the item selected in the left panel. For example, an item depends on another if it includes, calls, sets, uses, casts, or refers to that item.

You can expand files and architecture nodes 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. You can also right-click on either panel and choose **Expand All** or **Collapse All**.

The Dependency Browser has the following toolbar icons:



Click the **Filter** icon to enable or disable any of the dependency types shown. The options change depending on the source code language. You may want to hide “include” and “import” relationships if they are required for building but are not logically dependent. This option and an option to show either Compile Time or Link Time dependencies (for languages that make a distinction) can be controlled in the **Dependency** category of the **Tools > Options** dialog (page 93).



Click the **Graph** icon to create a graphical view of the dependencies currently shown in the Dependency Browser. Such graphs are available for architecture nodes,

classes, and files. While the Dependency Browser shows one level of dependency, graphical views can show multiple levels. This icon is enabled if the drop-down relationship is set to Depends On or Depended On By. It is not enabled for if the relationship is set to References From or References To. See page 201 for details.

Click the  **Export** 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. Press Ctrl+C to place the currently selected text on your clipboard for pasting into other applications. See page 139 for other ways to export information about dependencies.

Click the  **Favorites** icon to add the current dependency to your Favorites list. See page 142.

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

If the **Sync** box is checked, the Dependency Browser displays information about any architecture node, file, class, or package that you select in the Project Browser, Entity Filter, Architecture Browser, or similar window. In addition, the Dependency Browser displays information about any relationship (a connecting line or “edge” for which right-clicking the line shows a list of references) you select in a graph.

Click the hamburger menu  to change any of the following display options:

- **Architecture Name as.** The default is to show the relative name, but you can select short name or long name instead.
- **File Name as.** The default is the short name. You can select the relative name or long name instead.
- **Entity Name as.** The default is the short name. You can select the long name instead.
- **Group By:** Choose how you want to organize the dependencies listed in the right pane. By default they are grouped based on the top-level item in the left pane. For example, if the top-level item is a file, then the grouping is by file. Other options are None (flat list), By Dependency / Left Root, By Entity, By Class, By Definition File, and By *architecture\_name*. The Compile Time vs. Link Time setting for dependencies affects grouping only if you group by Dependency / Left Root (see page 104).
- **Nested Groups:** Choose whether to organize the dependencies listed in the right pane into nested groups. For example, dependencies may be grouped first by file and then by function. By default, such nested grouping is used only if a group contains more than 20 references, but you can choose to have no nested groups or to always use nested groups.
- **Architecture Tree:** Choose whether to organize the dependencies listed in the right pane using the architecture hierarchy. By default, nesting architecture nodes is used only if a node contains more than 20 references, but you can choose to have no nesting of nodes or to always nest nodes.
- **References Sorted By:** Choose whether to sort dependencies by Location (the default), Scope, Entity, or Kind.

## What are Dependencies?

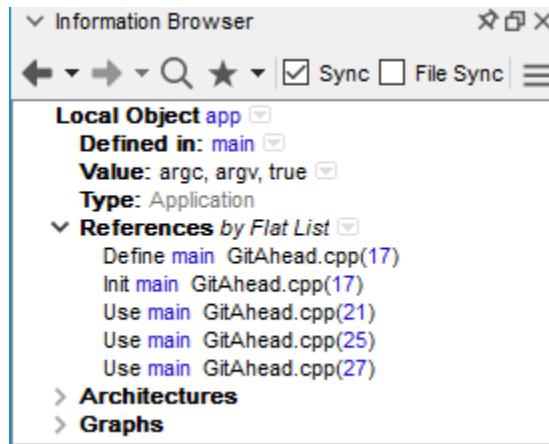
This section provides more detail about how *Understand* determines dependencies for display in the Dependency Browser and in dependency graphs.

A *reference* is a location in the code that connects two entities. For example, myClass may declare a member function myFunction. In this example, the reference connects a parent entity and a child entity. You can view references involving an entity in the Information Browser.

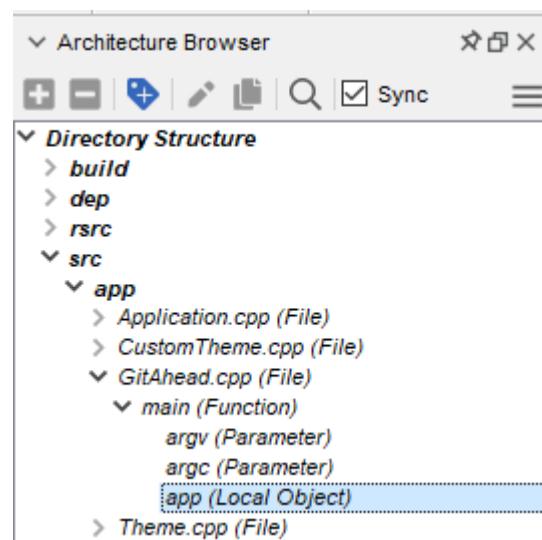
A *dependency* is a connection due to the reference(s) between two files, classes, architecture nodes, and certain other language-specific constructs. For example, file A depends on file B if file A needs file B to compile or link. Whether dependencies are based on compilation or linking is determined by your setting of the Compile Time vs. Link Time option (see *Dependency Category* on page 104).

Thus, dependencies are not between two entities. Instead, they are between the parents of groups of entities. Parents of entities may be files, classes, and/or architecture nodes. To find non-file parents of an entity, right-click and choose **Interactive Reports > API Info**. In the iReport area, see the “parent” property.

*Understand* determines the parent of each entity during analysis. In this Information Browser, you can see that the local object “app” is defined in main, so app and main have a parent-child relationship. The function main is a child of the file GitAhead.cpp. File entities have no parent entity.



Architectures organize entities into a hierarchy, which can extend the parent chain of entities. For example, the built-in “Directory Structure” architecture groups entities by folder. So, the full parent chain of the “app” object in the architecture is app (local object) -> main (function) -> GitAhead.cpp (file) -> app (architecture node) -> src (architecture node) -> Directory Structure (architecture). Custom architectures can organize non-file entities in addition to file entities. To see the parent chain of entities in the Architecture Browser, turn on **Show Implicitly Mapped Child Entities**.



*Understand* calculates dependencies at the file, class, and architecture level. Files can depend on other files, classes can depend on other classes, and architectures can depend on other architectures within the same root architecture. A reference is not considered a dependency if the levels of the entities involved do not match. So class dependencies don't include any references to non-member functions, and architectures don't include references to anything that isn't mapped within the root architecture.

The basic steps *Understand* uses to find a list of dependencies to display are:

- 1** Find all entities in the starting group.
- 2** Get the list of references for each entity.
- 3** Determine the group at the given level (file, class, or architecture) for the second (referenced) entity.
- 4** If the group of the second entity exists and is not the same as the starting group, then the reference is a dependency connecting the two groups.

There are a few things to be aware of when finding dependencies.

Entities may belong to multiple groups. When an entity belongs in multiple groups, such as belonging in both a definition file and a declaration file or belonging to two nodes in the same architecture, then the best group to show is chosen by the dependency calculation.

- For C/C++, the best group search uses the include tree.
- For C/C++ and similar languages, when deciding whether an entity belongs to the file in which it is declared or defined, your setting of the Compile Time vs. Link Time option is used (see *Dependency Category* on page 104). Thus, if you choose Compile Time Dependencies, an entity belongs to the header file where it is declared. If you choose Link Time Dependencies, an entity belongs to the source code file where it is defined.
- For all languages the best group can also be determined by distance to the starting group through the architecture hierarchy. The Directory Structure architecture is used for the file level.
- If the calculation finds multiple “best” groups, the reference will be used as a dependency to each of the best groups.
- For Ada code, there are certain special cases related to file dependencies. When Ada file dependencies are calculated, the reference kind impacts the dependent file. Call, instanceof, and separate from references depend on the body file. For packages, procedures, functions, and non-object tasks and protected units, the reference kinds rename, use, and typed also depend on the body file. Ada parent, usepackage, and with references depend on the spec file. Any other entity and reference types depend on the file determined by the parser.

## Exporting Dependencies

The **Project > Export Dependencies** menu command lets you export several types of files that provide metrics about dependencies between architectures, files, and classes/packages.

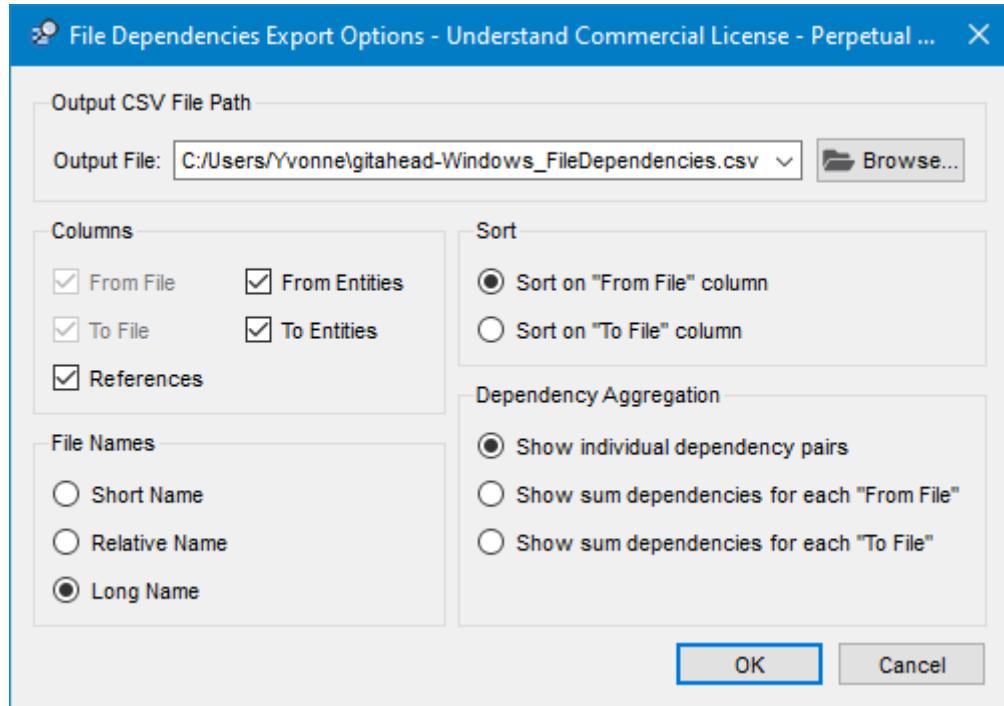
The output for most of these commands is in CSV (comma-separated values) format, which can be opened with most spreadsheet programs. When you create a CSV file with *Understand*, it is automatically opened in a text file window.

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 140.)
  - **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 141.)
  - **Export Cytoscape XML:** This output format can be opened in Cytoscape ([www.cytoscape.org](http://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 141.)
  - **Export Dot:** This output format uses a plain text graph description language. Various programs are available that can import DOT files to create graphs, perform calculations, and manipulate the data.
- **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 140.)
  - **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 141.)
  - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See page 141.)
- **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 140.)
  - **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 141.)
  - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See page 141.)

## 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.)

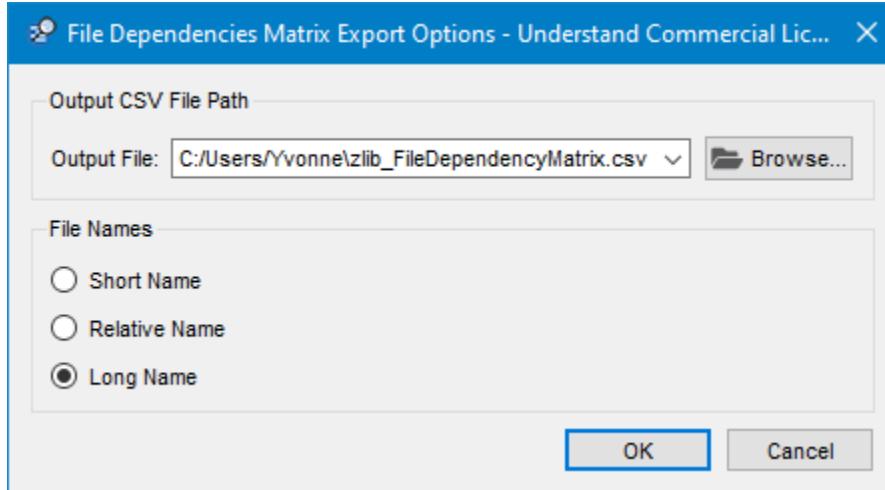


In this dialog, you can set the following options:

- **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](http://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.

## Favorites

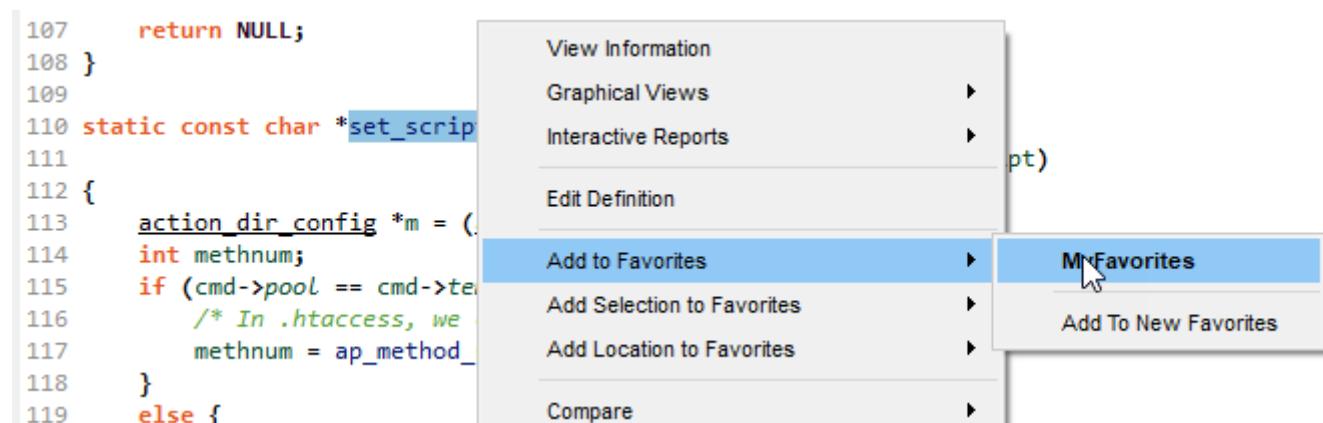
You can mark all kinds of things in *Understand* as “Favorites” so that you can quickly access them as you would web pages in a browser’s Favorites group. Your favorites can include entities, code locations, graphs, Information Browser displays, and dependencies. You can also store multiple plain text 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. If you want to mark code locations on a cross-project basis, see *Annotations* on page 186.

### Creating a Favorite Entity

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

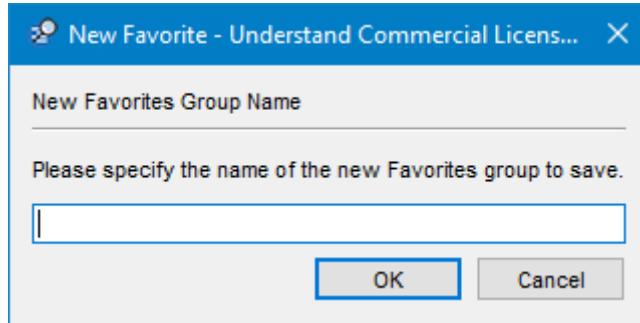
- 1 Select an entity name and right-click in source code, the Entity Filters area, the Information Browser, a graphical view, or anywhere else entities occur.



- 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 145 for information about the **Add Selection to Favorites** command, which stores text for pasting from the clipboard.

- 4 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**.



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

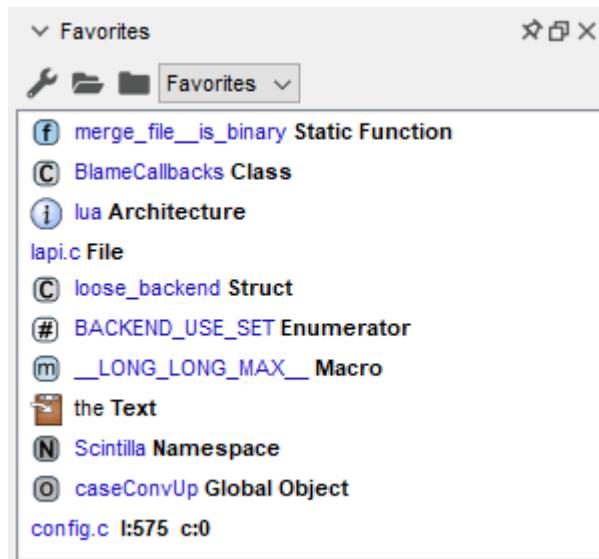
#### **Creating a Favorite View**

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 favorites group from the drop-down menu in the Favorites view toolbar.

#### **Using a Favorites Group**

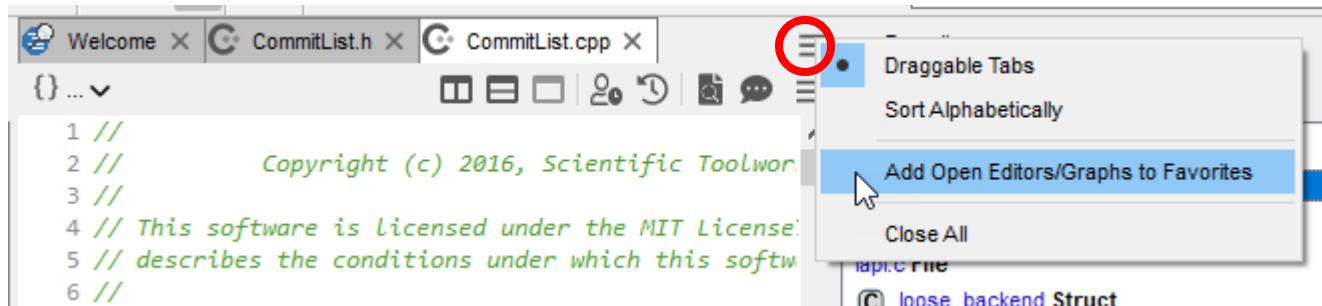
To open a Favorites group, choose **View > Favorites**.



In the Favorites view, you can use the drop-down list in the toolbar 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 all items** icon and close all the favorites in the list by clicking the **Close all items** icon.

You can add all the currently open files and graphical views to a Favorites list by clicking the upper hamburger menu 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, some of the icons and their meanings are as follows:

Favorites with an information icon link to an Information Browser view.

Favorites with a dependency icon link to a Dependency Browser view.

Favorites with a graph icon link to a graphical view.

Favorites with a clipboard icon store text that you can paste into source code. See *Creating a Plain Text Favorite* on page 145 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 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 145 for details.

Delete the selected favorite from the group. You can also delete a favorite by going to its location, right-clicking, and choosing the **Remove from Favorites** command.

Rename the current favorites group.

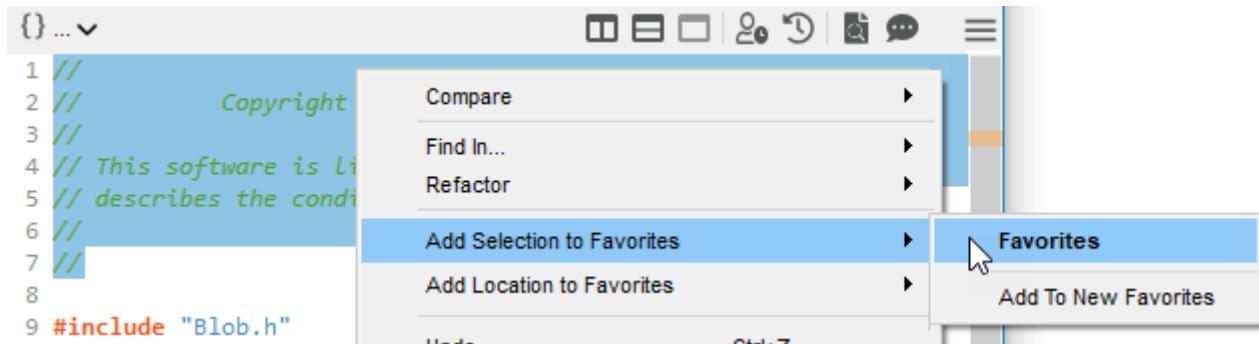
Delete the current group of favorites.

## 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.

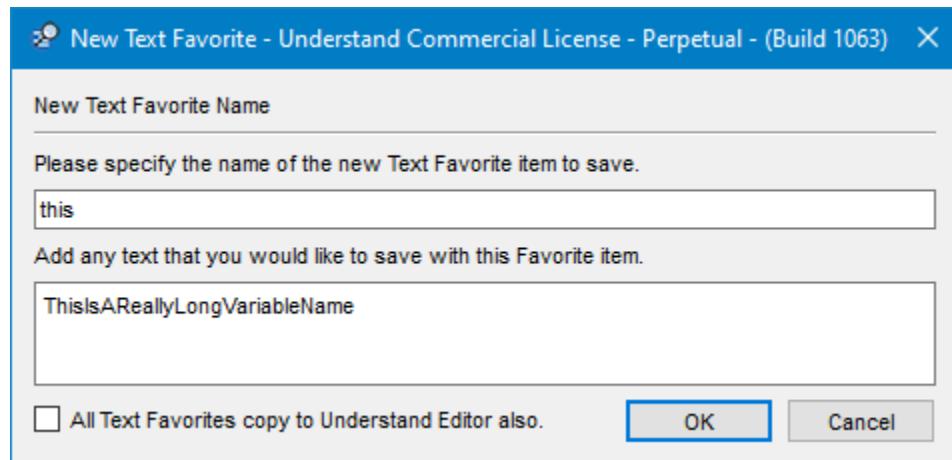
To save text as a favorite, follow these steps:

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



- 2 Choose **Add Selection to Favorites** and select a favorites group in 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.

---

## Chapter 6     **Searching Your Source**

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:

---

<b>Section</b>	<b>Page</b>
Searching: An Overview	147
Instant Search	148
Find in Files	150
Entity Locator	155
Finding Windows	159
Searching in a File	163

---

## 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 155	project-wide, entity-based
Find in Files	page 150	project-wide, text-based
Replace in Files	page 153	project-wide text-based
Instant Search	page 148	project-wide text-based
Find Next and Previous	page 163	single file
Find & Replace	page 163	single file
History	page 160	project-wide
Bracket Matching	page 180	single file
Favorites	page 142	project-wide
Contextual Information Sidebar	page 164	single file
Bookmarks	page 184	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 123 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.

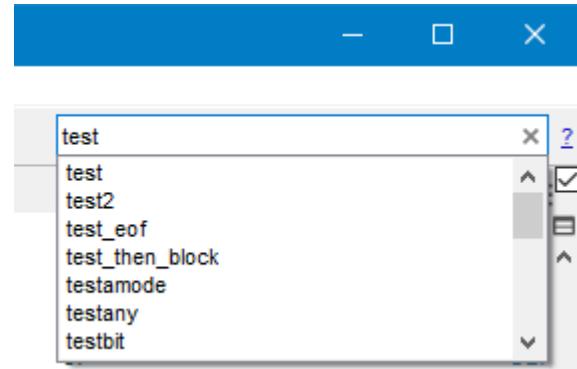
The search box for Instant Search is in the upper-right corner of the *Understand* window.

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. The Search Results interface for Instant Search is the same as the Search Results for Find in Files (see page 150).

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 screenshot of the Search Results interface. The title bar says 'Search Results'. Below it, the path 'E:\samplecode\zlib\example.c' is shown. The main area displays a list of 728 search results found in 46 files of 69 files searched. The results are as follows:

```

728 < Search for error: 728 results found in 46 files of 69 files searched. 2:57:35 pm (Less than a second)
10   > zlib\contrib\minizip\minizip.c
4     <--> zlib\contrib\minizip\lztools.c
1       88 err = Z_STREAM_ERROR;
1       131 err = Z_MEM_ERROR;
1       172 err = Z_STREAM_ERROR;
1       278 err = Z_STREAM_ERROR;
41   > zlib\contrib\minizip\unzip.c

```

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 [lucene.apache.org/core/3\\_6\\_2/queryparsersyntax.html](http://lucene.apache.org/core/3_6_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 parentheses to define the order of Boolean operators in searches. For example:

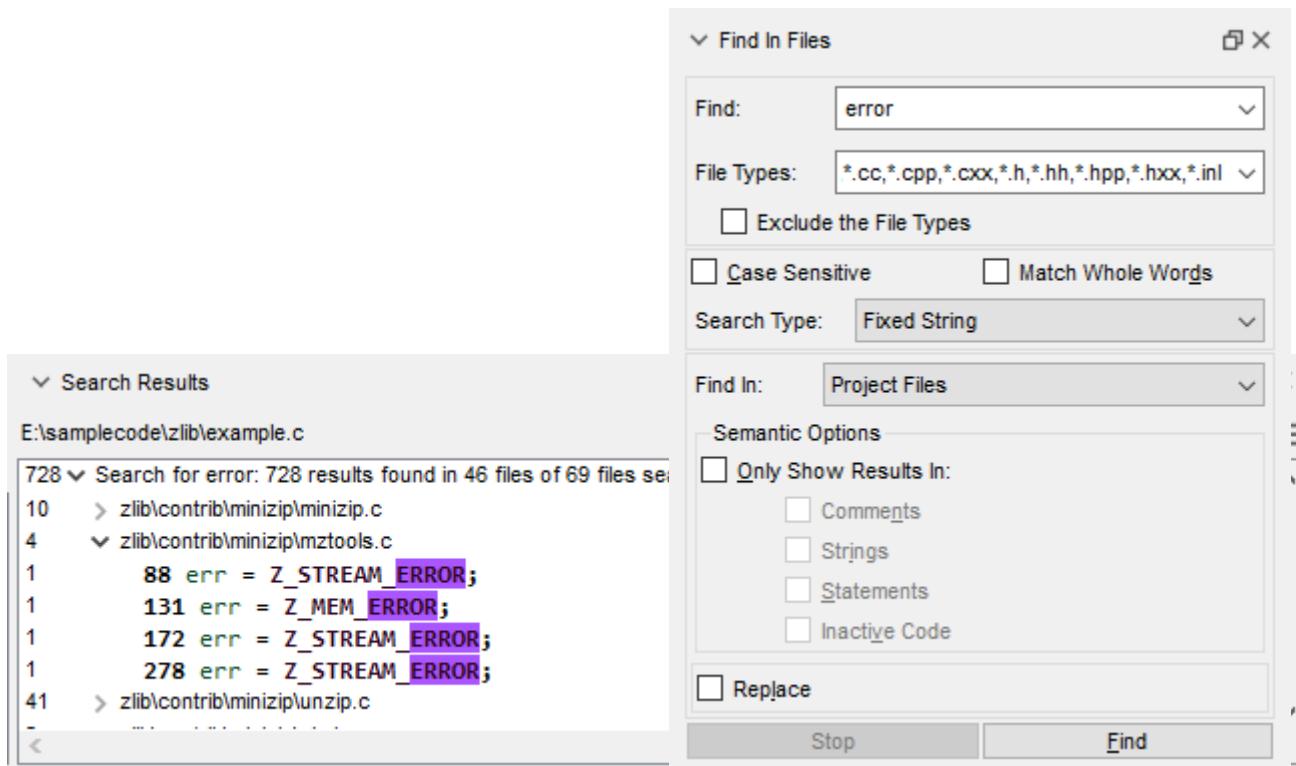
```
(delta0 OR delta1) AND change
```

- 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 *Search 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 153) 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.

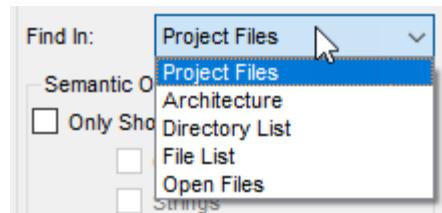


The Find in Files area allows you to search multiple files for the occurrence of a string. In previous versions, this feature was called Hyper Grep for its similarity to the Unix command *grep*. Specify a search as follows:

- **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 **Exclude the File Types** box 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 158 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.

For **Architecture, Directory List, and File List** searches, click + to add a location. Click the pencil icon to modify the selected location. Click the red X icon to delete the selected location. You can uncheck a location to temporarily disable searching it.



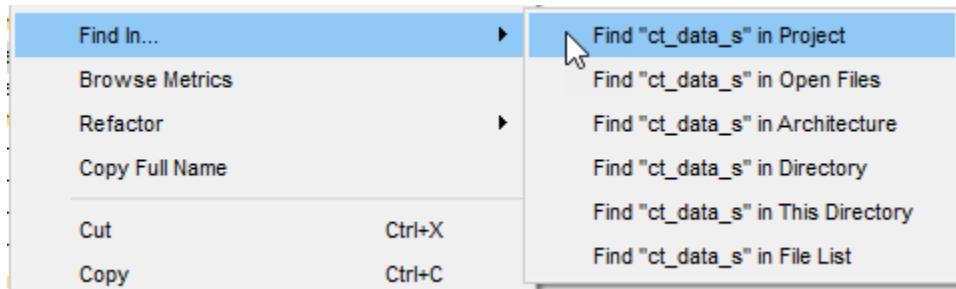
If you select **Architecture**, click + to browse for an architecture node.

If you select **Directory List**, click + to browse for directories. You can click the X icon to temporarily exclude the selected directory from the search or the trash can icon to permanently exclude a directory. Sort the list with up and down arrows.

If you select **File List**, you can click + to browse for files.

If you select **Open Files**, all files that are currently open are searched. You can check the **Search Active File Only** box or use the **File Types** specification to limit which open files 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 153) dialog by checking this box.

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

## Search Results

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

The screenshot shows a Windows-style search results window titled "Search Results". The path "E:\samplecode\zlib\example.c" is shown at the top. A message indicates "728 results found in 46 files of 69 files searched. 2:57:35 pm (Less than a second)". The results list contains several lines of code from the file, with specific error codes highlighted in purple: "88 err = Z\_STREAM\_ERROR;", "131 err = Z\_MEM\_ERROR;", "172 err = Z\_STREAM\_ERROR;", "278 err = Z\_STREAM\_ERROR;". The toolbar at the top right includes icons for Undo, Redo, Find, Copy, Paste, and a search bar.

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 Search 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 for the results provides the following controls:

- Undo and Redo action, for example the action to remove a search result from the list by selecting a row and clicking the icon on the right end of the row.
- Go to Find in Files dialog.
- Open the selected match in the Source Editor.
- Search within the currently selected set of results (using the same search bar described in *Searching the Information Browser* on page 128).

The hamburger menu in the upper-right corner provides the following commands:

- **Display Files As** shows Short Names (only the filename), Full Names (the full file path), or Relative Names (the file path relative to the project).
- **Organize Results By** changes the organization of the most recent results. The choices are a flat list with the filename next to each search result, a list grouped by file, or a tree using an architecture.
- **Show Criteria** displays the search options used for the search in the line that shows the number of results.
- **Expand All On Search** expands all lines when a new search is performed.
- **Clear Results Before Search** automatically clear the results of the previous search when you run a new search. You can use the Undo icon to restore cleared results.

From the right-click 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 Results window by choosing **Search > Show Search Results**.

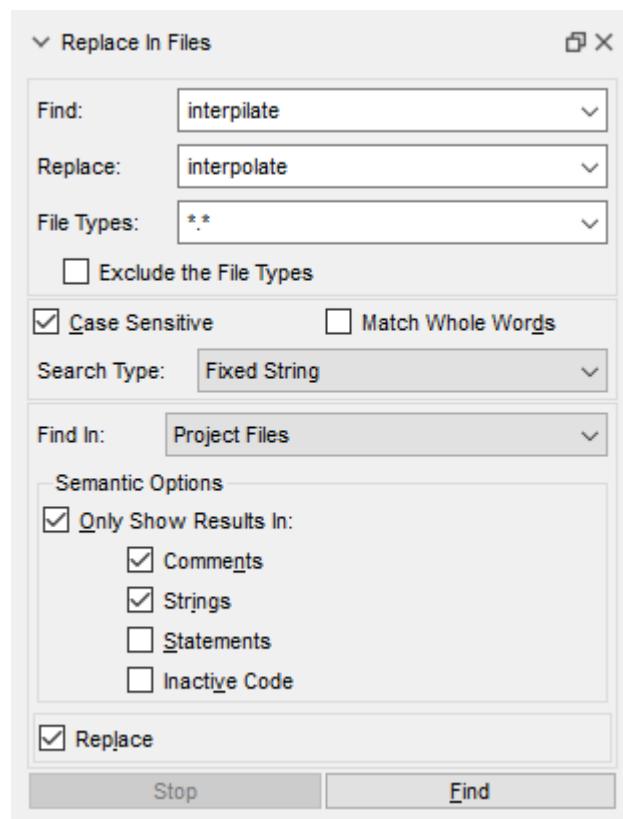
## Replace in Files

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 - Non Greedy” in addition to the options available for Find in Files.

To switch between the Replace in Files and Find in Files tools, check or uncheck the **Replace** box just above the Stop button.



*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.

The results are displayed in the same Search Results area whether you check or do not check the **Replace** box. But, if you checked the **Replace** box and typed a **Replace** string, you see the results with the text substitution.

```

3      < zlibcritical.c
1          11 const char inflatedecompress_copyright[] =
1          12 " inflatedecompress 1.2.3 Copyright 1995-2005 Mark Adler ";
1          32 int inflatedecompress_table(type, lens, codes, table, bits, work)
1      < zlibcritical.h
1          53 extern int inflatedecompress_table OF((codetype type, unsigned short FAR * v

```

Click on a result to open the Replace Preview view, which shows a comparison of the file that contains the result before and after the text is replaced. This is just a preview; the text has not yet been replaced.

The screenshot shows the 'Replace Preview' interface for the file 'zlib-thin.ads'. It displays two columns: 'Current' and 'Replaced'. The 'Current' column shows the original code, and the 'Replaced' column shows the code with the specified replacements applied. The interface includes navigation arrows and a status bar indicating '11/17' results.

```

389 pragma Import (C, DeflateEnd, "deflateEnd") 389 pragma Import (C, DeflateEnd, "deflateEnd")
390 pragma Import (C, Inflate, "inflate"); 390 pragma Import (C, decompress, "decompress")
391 pragma Import (C, InflateEnd, "inflateEnd"); 391 pragma Import (C, decompressEnd, "decompr")
392 pragma Import (C, deflateSetDictionary, " 392 pragma Import (C, deflateSetDictionary, "
393 pragma Import (C, deflateParams, "deflate 393 pragma Import (C, deflateParams, "deflate"
394 pragma Import (C, inflateSetDictionary, " 394 pragma Import (C, decompressSetDictionary,
395 pragma Import (C, inflateSync, "inflateSy 395 pragma Import (C, decompressSync, "decomp
396 pragma Import (C, inflateReset, "inflateF 396 pragma Import (C, decompressReset, "decom
397 pragma Import (C, compress, "compress"); 397 pragma Import (C, compress, "compress");

```

In the Search Results, replace text by right-clicking on a row and choosing **Accept** (Ctrl+K) or **Accept / Visit Next** (Ctrl+Alt+P). Reject a replacement by right-clicking and choosing **Dismiss Result(s)** (Del). If you choose these commands when right-clicking on a filename or directory structure node, the commands apply to the entire file or all files in the directory and its children.

When you accept or reject replacements, they are removed from the list of results, since you have already dealt with them.

Right-click and choose **Undo** if you want to cancel the most recent replacement.

The screenshot shows the 'Search Results' interface. It displays a tree structure of search results. A specific result in 'zlib\contrib\ada\buffer\_demo.adb' is highlighted with a blue selection bar. The interface includes navigation arrows and a status bar.

```

20      > zlib\zlib.h
270      < contrib
43      < ada
1        < zlib\contrib\ada\buffer_demo.adb
1          80 Inflatedecompress_Init (Decompressor);
1        < zlib\contrib\ada\test.adb
1          329 ZLib.Inflatedecompress_Init (Filter, Header => Header);
31      < zlib\contrib\ada\zlib-thin.ads
1          131 function InflatedecompressEnd (strm : Z_Stream) return Int;
1          150 function InflatedecompressSetDictionary

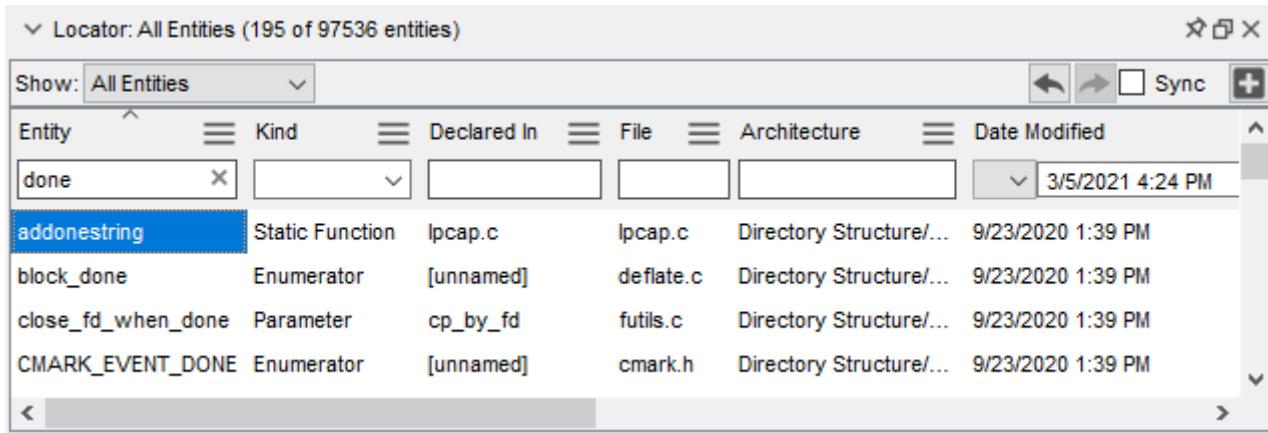
```

Other right-click commands in the Search Results area allow you to expand and collapse the results tree, copy results, and navigate to the previous or next result, file, or set of results from another search.

## 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 project. 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.

If you check the **Sync** box, selecting entities in other *Understand* windows causes the Entity Locator to select that entity unless filters prevent the entity from being displayed.

### 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.

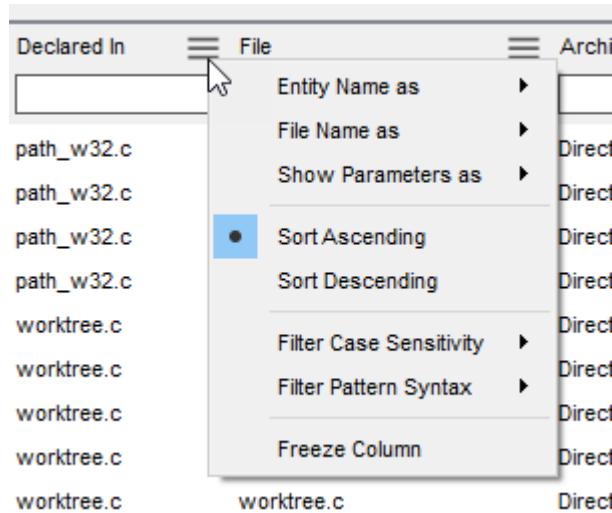
You can right-click on a column header and choose **Freeze Column** to move that column to the left and prevent it from being repositioned.

### Long versus Short Names

In the *Entity*, *Declared In* and *File* columns, you can right-click the column header or click the hamburger menu and choose **Entity Name as** to set the display format for entity names and **File Name as** to set the display format for 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**

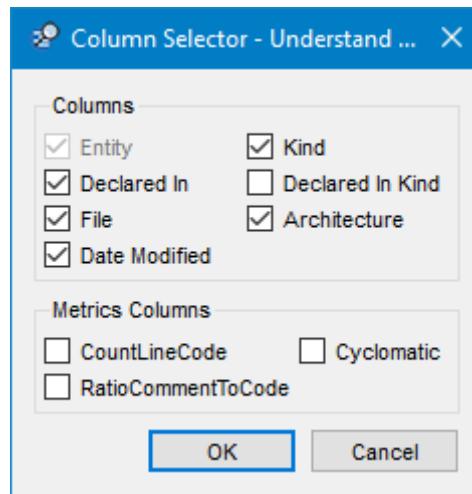
Column headers are tools in the *Entity Locator*. Left-click them to sort according to that column. Right-click a column or click the hamburger menu to see commands that let you control how entities are listed, sorted, and filtered.



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 + 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.

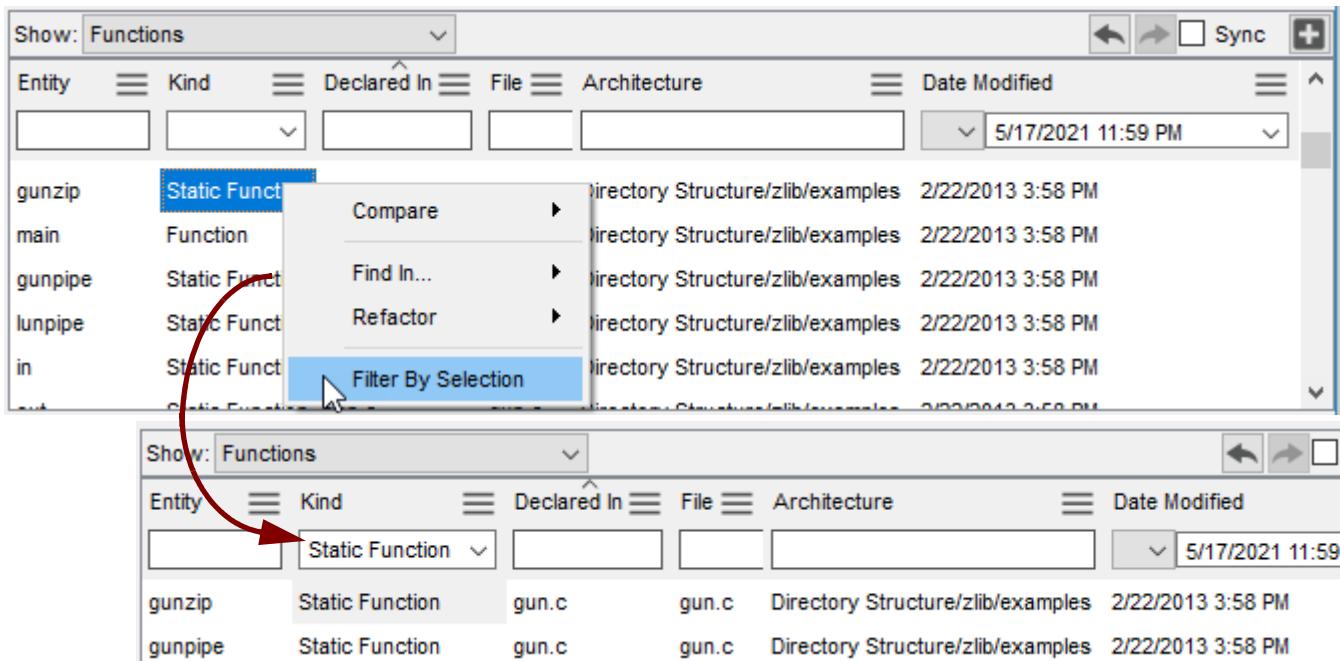
For example, you may filter by the *Kind* column selecting a kind from the drop-down list. You can also right-click on any item listed in the *Kind* column and select **Filter By Selection** from the menu. This filters the list of entities to contain only entities of the kind you selected. The title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

Or, you can simply type a filter in any of the fields. To search for field values that do not contain a particular string, type ! (exclamation mark) and then the filter.

To clear a filter, just delete the text from the field in the column heading or right-click a column header and choose **Clear Filter** or **Clear All Filters**.

You can use the **Previous** and **Next** buttons to move through the history of filters you have used.

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 ( $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$ ), 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.

Date Modified
$\leq$ <input type="text" value="6/3/2021 6:04 PM"/> $\geq$
5/21/2021 2:08 AM

Similarly, the metrics columns, which you can enable by *Choosing Columns* on page 156, 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 a hamburger menu 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 set the **Filter Pattern 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.

Using ! to search for field values that do not contain a particular string can be used with any Filter Pattern Syntax.

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

Symbol	Description	Example
^	Match at the beginning of a line only.	<code>^word</code> Finds lines with word starting in the first column.
\$	Match at end of a line only.	<code>word\$</code> Finds lines that end with “word” (no white space follows word).
\<	Match at beginning of word only.	<code>\&lt;word</code> Finds wordless and wordly but not fullword or awordinthemiddle.
\>	Match at end of word only.	<code>\&lt;word</code> Finds keyword and sword but not wordless or awordinthemiddle.
.	A period matches any single character.	<code>w.rd</code> Finds lines containing word, ward, w3rd, forward, and so on, anywhere on the line.
*	Asterisk matches zero or more occurrences of the previous character or expression.	<code>word*</code> Finds word, wor, work, and so on.
+	Match one or more occurrences of the previous character or expression.	<code>wor+d</code> Finds word, worrd, worrrd, and so on.
?	Match zero or one occurrences of the previous character or expression.	<code>wor?d</code> Finds word and wod.

Symbol	Description	Example
[ ]	Match any one of the characters in brackets but no others.	[AZ] Finds any line that contains A or Z.  [Kk][eE][Nn] Finds any variation of case when spelling "Ken" or "KEn" or "keN".
[^ ]	Match any character except those inside the brackets.	[^AZ] Finds any line that does not contain the letters A or Z.
[ - ]	Match a range of characters.	[A..Z] Finds any line containing letters A through Z on them but not lower case letters.
	A vertical bar acts as an OR to combine two alternatives into a single expression.	word   let+er Finds word, ilter, letter, lettter, and so on.
\	Make a regular-expression symbol a literal character.	\*\$/ Allows searching for *. This example finds all lines ending in */

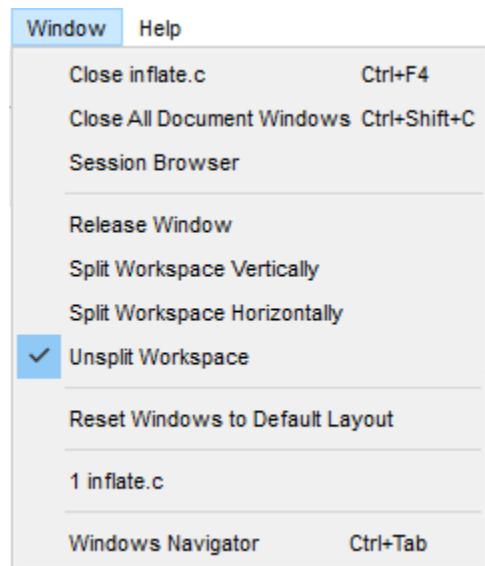
A full explanation of regular expressions is beyond the scope of this manual. Unix users may refer to the manual page for *regex* using the command “*man -k regex*”. For a comprehensive explanation, we refer you to the book “*Mastering Regular Expressions*”, published by O'Reilly and Associates ([www.ora.com/catalog/regex](http://www.ora.com/catalog/regex)).

## Finding Windows

If you have several windows open, you can use options in the **Window** menu 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, 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**.

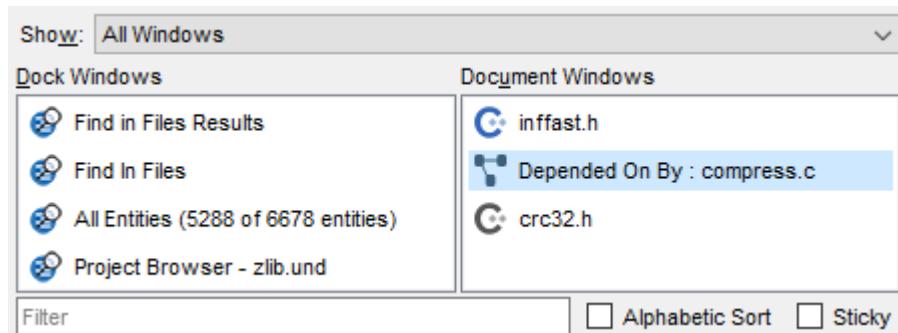
Choose **Window > Release Window** to change the tabbed area to a separate window that can be resized and moved around your screen. Select **Capture Window** from the hamburger menu  in the upper-right corner of a released window to replace the window within the main window.



The Window menu also lets you use **Window > Split Workspace Vertically** or **Window > Split Workspace 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 Workspace** to remove the split.

Use **Window > Reset Windows to Default Layout** to return to the original layout.

The **Window > Windows Navigator** command (Ctrl+Tab) opens a temporary list of currently open windows. When you press Enter or double-click on an item in this list, the list goes away, and focus is given to the selected item. Press Tab while the Windows Navigator is open to cycle through the list of items. 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 type in the **Filter** box.

Checking the **Alphabetic Sort** box sorts both the docked windows and the document windows.

By default, if you press Ctrl+Tab, the Windows Navigator closes and switches to the selected window when you release the Ctrl key. (You can press Tab repeatedly while holding down the Ctrl key.) You can check the **Sticky** box to cause the Windows Navigator to wait for you to press Enter or double-click.

---

## 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.



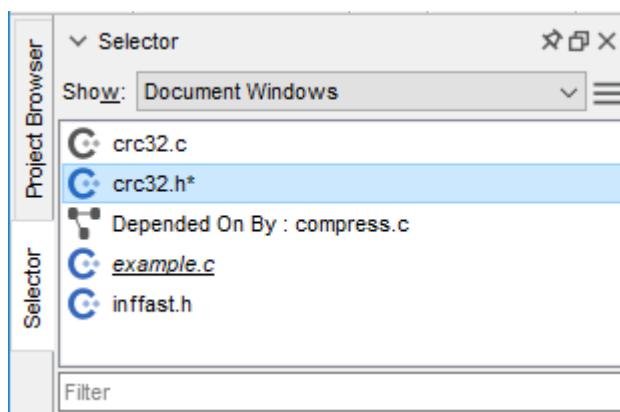
You can click the down-arrows to see the list of source locations in the history. You can choose **Clear History** from the drop-down Go Back list to clear the browsing history.

## View Menu Commands

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

The **View > Window Selector** command opens an area that lists currently open windows. Click a window name to make it active.

By default, this area lists only document windows, but you can use the **Show** dropdown to change the type of window listed. Any released windows are listed in underlined italics. The icons indicate the type of window, including whether the source file is unsaved.



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

You can use the hamburger menu to change the order from alphabetic to most recently used or by file extension. You can also use the hamburger menu to change the filename format to show the short, relative, or absolute file paths.

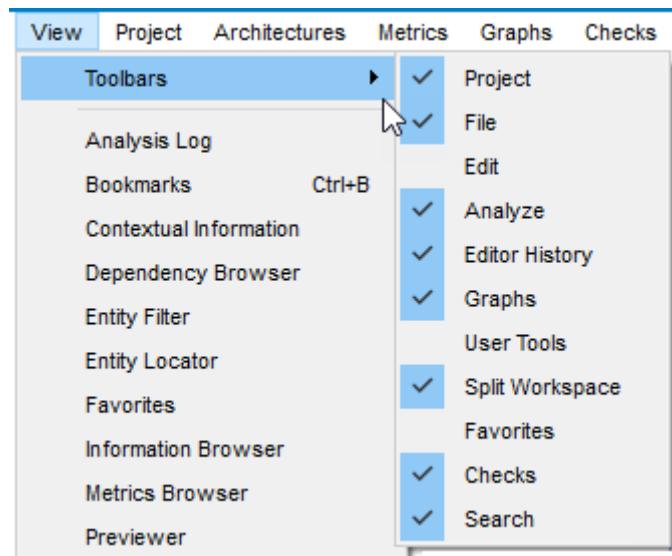
Using the Selector is a convenient way to perform actions—such as Close—on multiple windows by selecting multiple windows from the list, right-clicking, and choosing **Close Selected Window(s)** or **Close Unselected Window(s)**.

If you have created bookmarks in your source code (page 184), you can use the **View > Bookmarks** command to open the list of bookmarks.

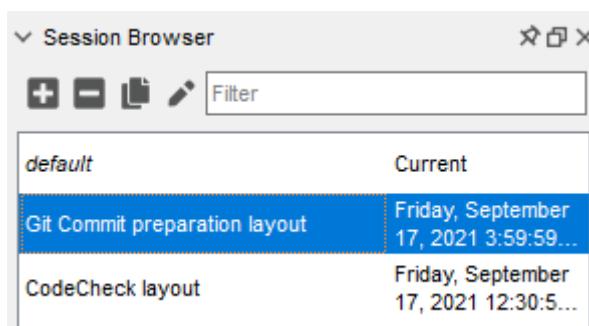
**Displaying Toolbars**

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 icons by choosing **View > Toolbars**.

**Session Browser**

*Understand* opens a project using the most recent session layout. Changes to which tools, toolbars, tabs, and other features are open and their sizes and locations are saved automatically as part of the session. You can create and save multiple session layouts using the Session Browser (page 162). Choose **Window > Session Browser** to see the list of sessions. You can use the toolbar icons for this area to add the current layout as a session, remove the currently highlighted session from the list, copy the highlighted session so that you can modify the layout from there, rename the highlighted session, and filter the list of sessions by name.



To return to the original layout, choose **Window > Reset Windows to Default Layout**.

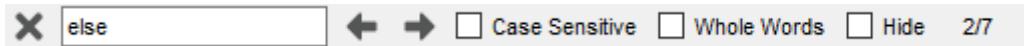
Session information is stored in the `session.json` file in the project folder and may be shared with other users.

## 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. If the string does not exist in the file, the search field turns red.

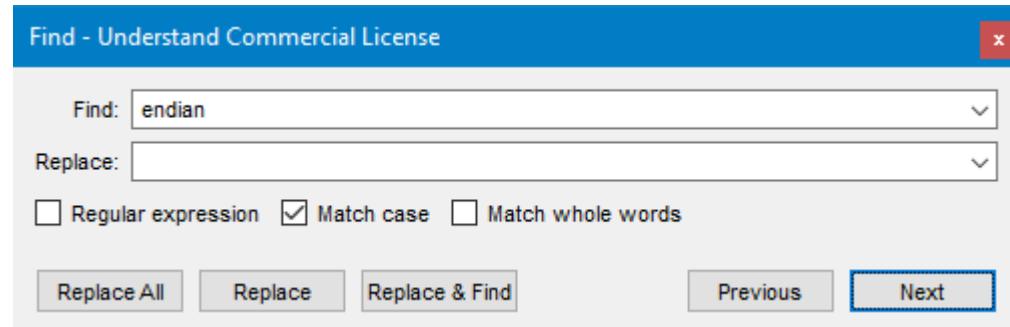
Use Ctrl+F to move to the next occurrence. Use Ctrl+Shift+F to find the previous occurrence.

Click the **Previous** or **Next** arrow to move from match to match. You can check the **Case Sensitive** and **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.

### Find & Replace

If you want to use Search-and-Replace or regular expressions for searching within a single source code file, 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**, or **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 157.

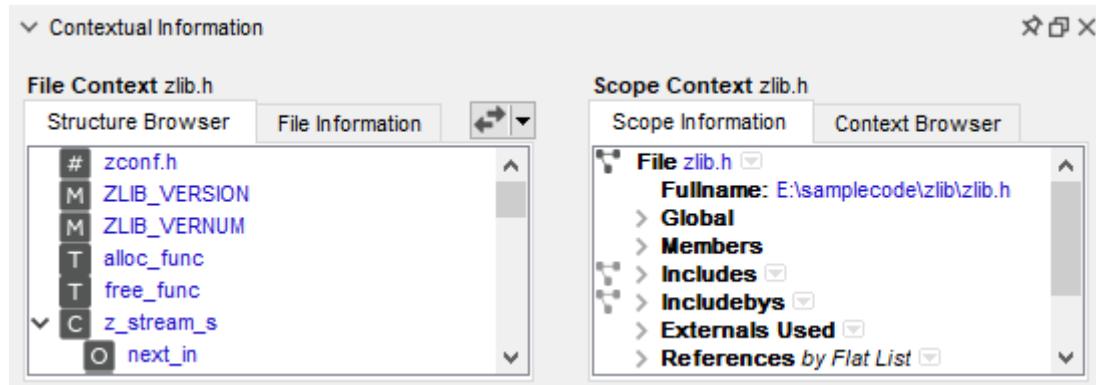
If you want to replace the string you are finding, type that in the **Replace** field.

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 within the current file.

The Find dialog searches only individual files. To search multiple files, see *Find in Files* on page 150.

## Contextual Information

The Contextual Information view is similar to the Scope List (see page 167), but more powerful. You can open this view by choosing **View > Contextual Information** from the menus. You can click the  icon in the Source Editor toolbar to open or close the Contextual Information view.



This view shows the structure of the currently active Source Editor. The tabs 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 to the right of the File Information tab changes the current file in the Source Editor and the Contextual Information view 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 7     Editing Your Source

This chapter covers *Understand*'s source and text file editor.

This chapter contains the following sections:

Section	Page
Source Editor	166
Saving Source Code	171
Refactoring Tools	172
Other Editing Features	179
Annotations	186
Printing Source Views	191

## Source Editor

The **Source Editor** offers a full featured source code editor, with syntax coloring and right-click access to information on most entities in your code.

```

479     char * submatch(file, pat, str, strend, k, altindex)
480         char file[], pat[], str[];
481         register char *strend, *k;
482         int altindex;
483     {
484         register char *s;
485         char *t, c;
486
487         t = k;
488         s = ((altflag) ? k - altlen[altindex] + 1 : k - altmin + 1);
489     #ifndef NOKANJI
490         c = ((altflag) ? altpat[altindex][0] : pat[0]);
491         if (c & NONASCII)
492             if ((s = kanji(str, s, k)) == NULL)
493                 return (++k); /* reject false kanji */
494     #endif
495         do;
496         while (*s != NL && --s >= str);
497         k = s + 1; /* now at line start */
    
```

CodeChecked: 8/22/2021 3:53 PM | Line: 477 Column: 19 | Tab Width: 8 | RW | C

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.

The **Editor > Advanced** category of the Understand Options dialog (see page 107) lets you control the format when you print from the Source Editor, highlights in the scroll bar when you search, copy-paste behavior, auto-completion, auto-indenting, and more.

The **Editor > Styles** category of the Understand Options dialog (see page 111) lets you change the colors and styles used for different types of source code. The **Key Bindings** category (see page 101) shows a list (and lets you modify the list) of keystrokes you can use in the Editor.

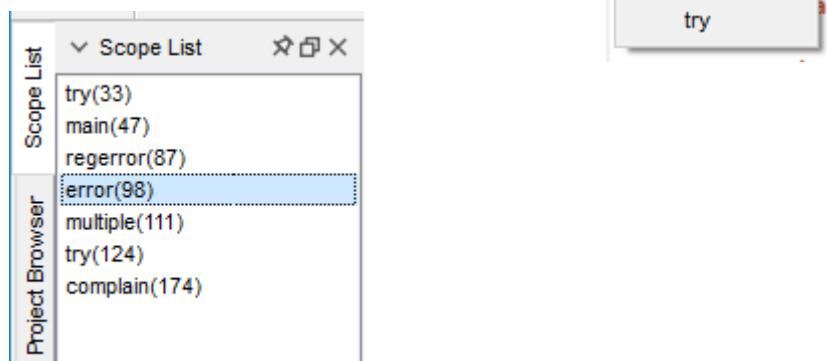
### File Icons

Each file in a Source Editor has an icon in its tab. The letter in the icon indicates the language used in the file. If the file has been modified but not saved, the icon is blue and an asterisk is shown next to the filename.

## Scope List

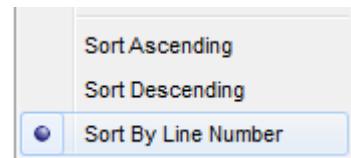
Scope lists are useful for jumping around in large files. You can jump to a particular function, procedure, or other language-specific construct in the current source file by selecting from the scope drop-down list in the toolbar. Constructs are listed alphabetically. The drop-down list shows all such constructs in the file the last time the project was analyzed.

You can choose **View > Scope List** to open the list in a Scope tab in the area where the Entity Filter is shown. The line number where the construct begins is also shown.



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.

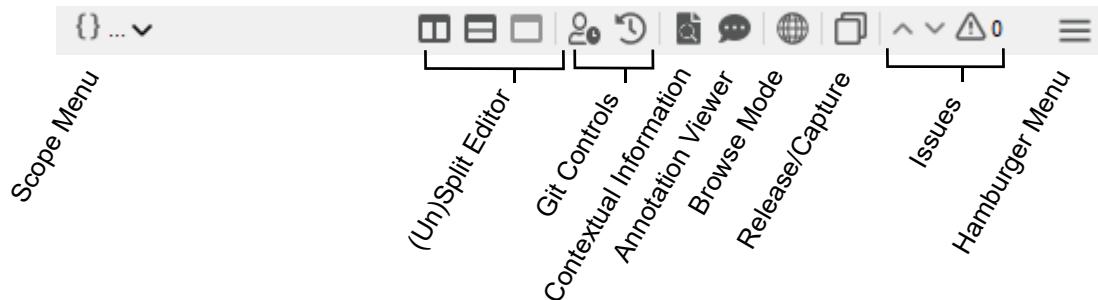
You can right-click on the Scope tab to choose a sort order from the context menu. The ascending and descending orders sort alphabetically or reverse alphabetically. The default is to sort by line number.



For more power than the scope list, use *Contextual Information* on page 164.

**Toolbar**

The toolbar for a Source Editor may contain the following icons:

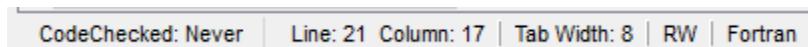


You can hide or display all of the icons in the Source Editor toolbar using the **Toolbar Sections** category in the hamburger menu. The toolbar sections available are:

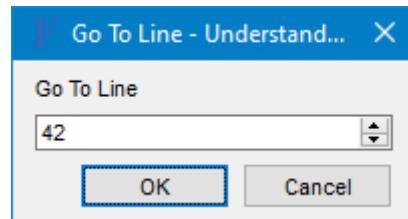
- **Scopes:** Select a function or similar entity to jump to.
- **Split Editor:** Split the Source Editor window horizontally, vertically, or remove a split.
- **Version Control:** Perform Git integration actions. Show/hide the blame column in the margin and open the uncommitted changes differences view. See *Exploring Git History* on page 267 for details.
- **Dock Windows:** Show/hide the Contextual Information view and the Annotations viewer.
- **Released Windows:** Release/capture window within the main window.
- **Issues:** Move to next/previous issue found in source code.

**Status Line**

When a Source Editor is the active window, the status bar at the bottom of the *Understand* window shows the last time CodeCheck was run on this file, the line number and column number of the cursor position, the tab width setting for this file, whether the file is in read-write or read-only mode, and the source language.



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 Tab Width, you can change the tab width for this file. See page 182.

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 this file is treated as using.

## 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 drag the mouse to 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.

```

43      */
44      #include <stdio.h>
45      #include <cctype.h>
46      #include <sys/types.h>
47      #include <sys/stat.h>
48      #include "regexp.h" /* must

```

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.

For entities with a class path and files, the **Copy** command copies the short name. The **Copy Full Name** command in the context menu copies the full class path or file path.

## 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.

See page 112 for settings to control the behavior of Browse Mode.

## Context Menu

The context menu in the Source Editor provides a number of exploration and editing features. Many features let you find 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 126)
- Graphical Views (see Chapter 10)
- View Dependencies (see page 135)
- Interactive Reports (see page 284)
- Add to Favorites (see page 142)
- Compare (see page 262)
- Analyze (see page 88)
- Add/Remove to Architecture (see page 197)
- Annotate (see page 186)
- Explore (see page 134)
- Find in... (see page 150)
- Browse Metrics (see page 208)
- Hide Inactive Lines (see page 180)
- Add Bookmark (see page 184)

The following *editing features* are also typically included in the context menu:

- Refactor (see page 172)
- Undo / Redo
- Cut / Copy / Paste (see page 169)
- Copy Full Name (see page 169)
- Select All (see page 169)
- Remove from Project (see page 44)
- Edit Files (see page 129)
- Jump to Matching Brace (see page 180)
- Select Block (see page 180)
- Hide/Show Inactive Lines (see page 180)
- Fold All (see page 180)
- Soft Wrap (see page 182)
- Comment Selection / Uncomment Selection (see page 182)
- Sort Selection (see page 182)
- Change Case (see page 181)
- Reindent Selection/File (see page 182)
- Revert (see page 171)

**Hover Text**

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.

```
mTable = new QTableView(this);
mTable->verticalHe name: QTableView::QTableView );
mTable->setEditTri parameters: (none)
mTable->setSelecti returns: void View::SelectRows();
mTable->setSelectionMode(QAbstractItemView::ExtendedSelection);
```

## 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 since the last save, right-click in a file and choose **Revert**.

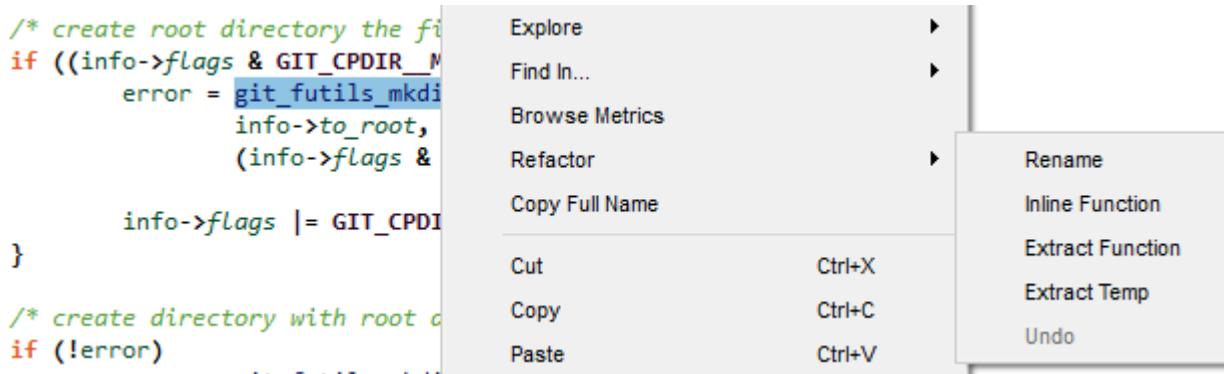
You can close the current source file by choosing **Window > Close <current\_file>** from the menus. You can also middle-click on the tab above the source file area to close that tab (if your mouse has a middle button).

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**.

## Refactoring Tools

Refactoring tools allow you to make structural changes to your code. Ideally, refactoring does not change the behavior of the code.

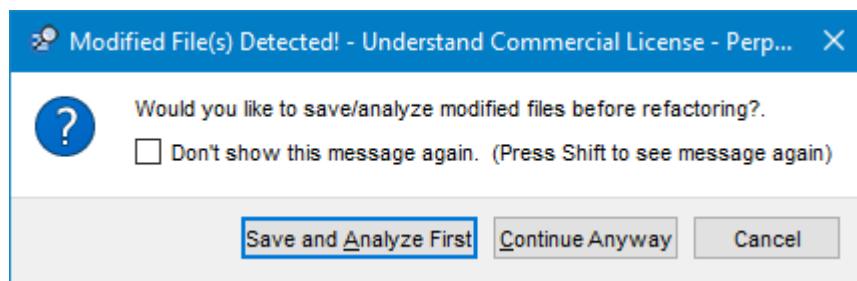
The refactoring tools allow you to preview the changes using a code comparison window. Refactoring changes can have significant effects on code, and should be reviewed before committing to make certain that the changes are correct.



The following refactoring tools are provided:

- **Rename:** page 173 (various languages)
- **Inline Function:** page 174 (C/C++)
- **Extract Function:** page 175 (C/C++)
- **Inline Temp:** page 176 (C/C++)
- **Extract Temp:** page 178 (C/C++)

If files have not been saved when you select a refactoring command, you are asked if you want to save the files and reanalyze the project. We recommend that you do this, so that the *Understand* project analysis will contain current information about your project.



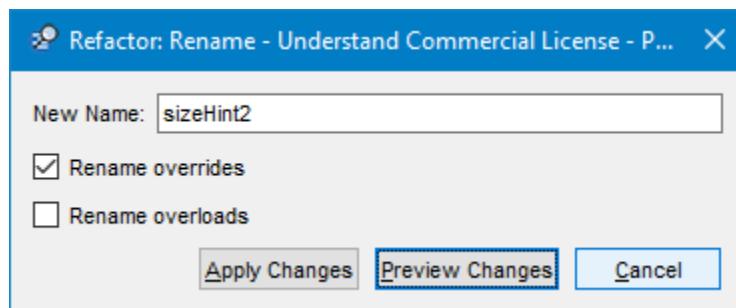
If you perform a **Refactor** operation and then decide that you did not want that change, right-click and choose **Refactor > Undo** from the context menu. In some cases, the refactoring operation cannot be undone because of subsequent changes.

## Renaming Entities

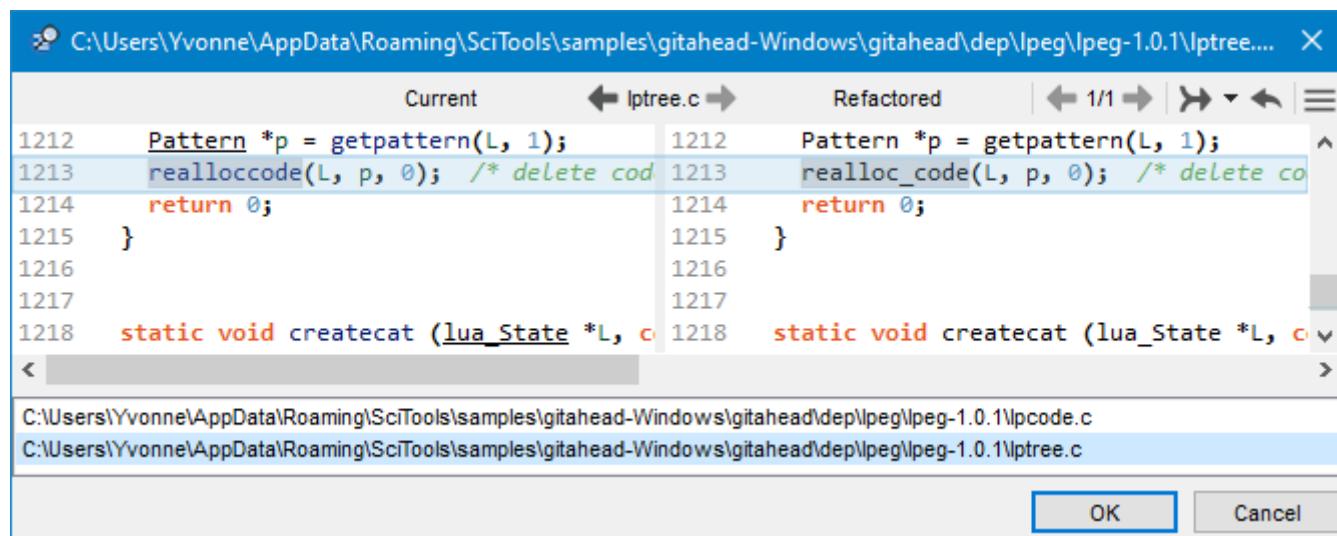
A command to rename entities is provided in order to support refactoring of your code in order to make your code more readable. It allows you to change the name of an entity throughout your project. The **Refactor > Rename** command is similar to **Replace in Files** (page 153). However, the difference is that the **Refactor > Rename** command determines which uses apply to that specific entity. This makes it a better way to rename such things as variables that are used locally, entities in applications with a variety of namespaces, and entities with names that may be a part of another entity name (for example, renaming a “src” variable without renaming “srcTimer”).

To use the **Refactor > Rename** command, follow these steps:

- 1 Highlight the name of the entity (for example, a function or argument) to change.
- 2 Right-click and choose **Refactor > Rename** from the context menu.
- 3 Type a new name for this entity in the dialog and click **Preview Changes**. If you are absolutely sure you want to perform the renaming operation, click **Apply Changes**. If the entity you are renaming is a C++ override, you can choose whether to rename overrides and whether to rename overloads.



- 4 A dialog opens that lets you examine all the instances where this entity name is used and how it will be changed. Exploring Differences on page 269 describes the icons and drop-down menus in this dialog.



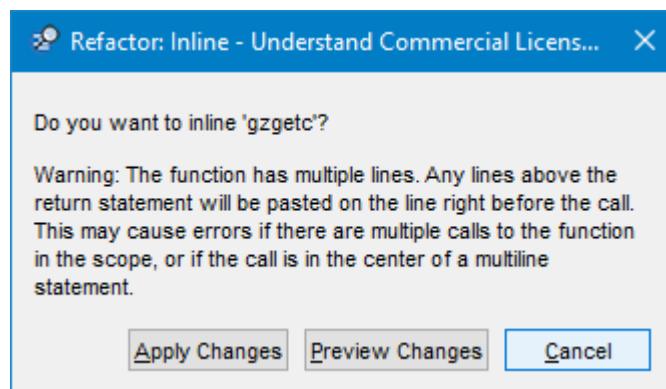
- 5 If you are sure you want to make all the changes, click **OK**.

## Inlining Functions

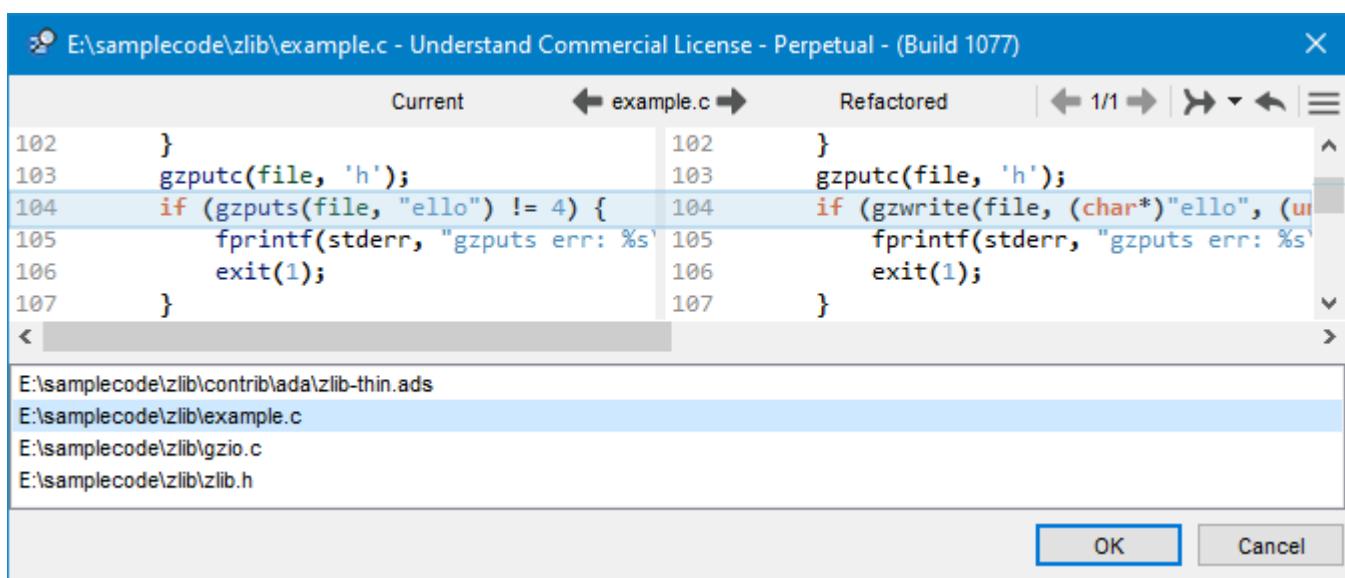
Inlining functions is a common optimization technique that places the code of a function at the location where it is called instead of in a separate function. In compiled languages, this can often be performed through compiler optimization, but inlining the source code may be useful for various reasons, including code clarity. Note that inlining involves tradeoffs. If a function is called in many places, inlining the code results in a larger code size and less maintainable code.

To use the **Refactor > Inline Function** command, follow these steps:

- 1 Highlight the name of the function you want to inline.
- 2 Right-click and choose **Refactor > Inline Function** from the context menu.
- 3 Click **Preview Changes**. If you are absolutely sure you want to perform the inlining operation, click **Apply Changes**.



- 4 A dialog opens that lets you examine all the changes that will occur. Exploring Differences on page 269 describes the icons and drop-down menus in this dialog.



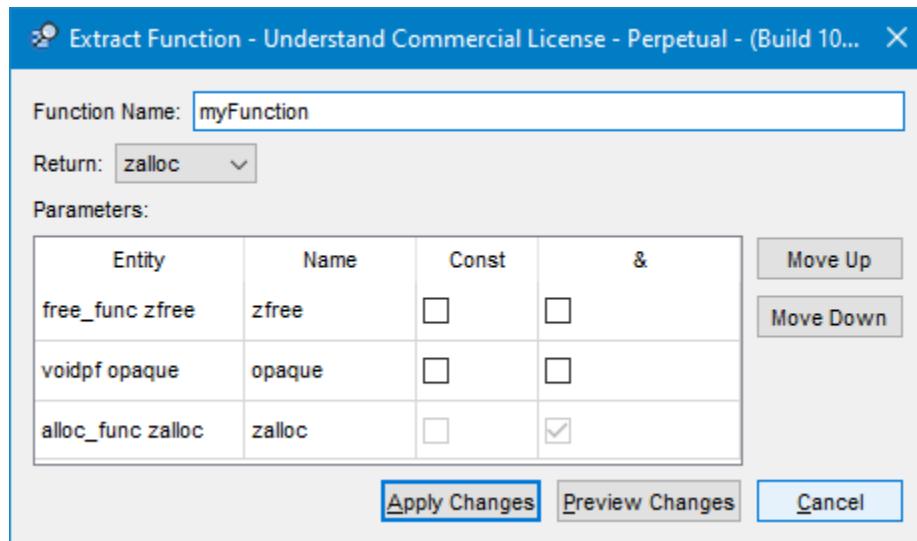
- 5 If you are sure you want to make all the changes, click **OK**.

## Extracting Functions

The opposite of function inlining is function extraction. You can extract some code to a separate function so that it can be called in several places and maintained in one place.

To use the **Refactor > Extract Function** command, follow these steps:

- 1 Highlight the code that you want to extract as a function.
- 2 Right-click and choose **Refactor > Extract Function** from the context menu.



- 3 Type a name for the function to be extracted and called from this code location.
- 4 Select the value or variable to be returned by the function.
- 5 For the parameters to be passed to the function, you can use the Move Up and Move Down buttons to change the sequence and check the boxes to identify parameters to be passed as const values or by reference.
- 6 Click **Preview Changes**. If you are absolutely sure you want to perform the operation, click **Apply Changes**.

- 7 A dialog opens that lets you examine all the changes that will occur. Exploring Differences on page 269 describes the icons and drop-down menus in this dialog.

The screenshot shows a Qt Refactoring dialog with two tabs: 'Current' and 'Refactored'. The 'Current' tab displays the original code for `GitCredential.cpp`, while the 'Refactored' tab shows the modified code. The 'Refactored' code includes a new function `processOutString` and calls it from the `store` method. The dialog also shows the file paths `D:\temp\gitahead-Windows\gitahead\src\cred\GitCredential.cpp` and `D:\temp\gitahead-Windows\gitahead\src\cred\GitCredential.h` at the bottom, along with 'OK' and 'Cancel' buttons.

```

74     password = value;
75 }
76 }
77
78     return !username.isEmpty() && !password.isEmpty();
79 }
80
81     bool GitCredential::store(
82         const QString &url,
83         const QString &username,
84         const QString &password)
85 {
86     QProcess process;
87     process.start(command(), {"store"});
88     if (!process.waitForStarted())
89         return false;
90
91     QTextStream out(&process);
92     out << "protocol=" << protocol(url) << endl;
93     out << "host=" << host(url) << endl;
94     out << "username=" << username << endl;
95     out << "password=" << password << endl;
96     out << endl;
97
80
81     void GitCredential::processOutString(const QString &password, QProcess &process, const QString &url)
82     QTextStream out(&process);
83     out << "protocol=" << protocol(url) << endl;
84     out << "host=" << host(url) << endl;
85     out << "username=" << username << endl;
86     out << "password=" << password << endl;
87     out << endl;
88 }
89
90     bool GitCredential::store(
91         const QString &url,
92         const QString &username,
93         const QString &password)
94 {
95     QProcess process;
96     process.start(command(), {"store"});
97     if (!process.waitForStarted())
98         return false;
99
100    processOutString(password, process, url);
101
102    process.closeWriteChannel();
103    process.waitForFinished();

```

- 8 If you are sure you want to make all the changes, click **OK**.

### Inline Temp

Inline temp refactoring can be used with a local or temporary variable that is initialized and never set after that. The inlining replaces uses of that variable with the expression to which it is initialized. In the following example, `patlen` could be inlined as `altmin`, so long as the value of `patlen` and `altmin` do not change between the initialization and any usage of `patlen`:

```

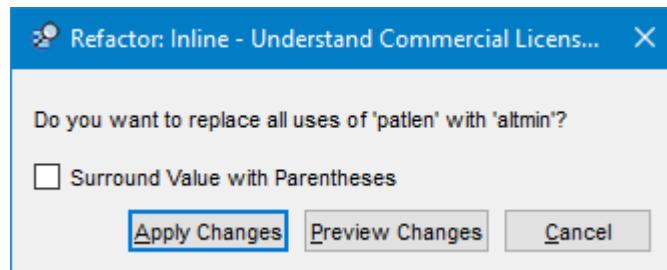
int patlen = altmin;

for (k = str + patlen - 1; k < strend;) {
    ...
}

```

To use the **Refactor > Inline Temp** command, follow these steps:

- 1 Highlight the variable to inline.
- 2 Right-click and choose **Refactor > Inline Temp** from the context menu.
- 3 If you want the expression that replaces the variable to be surrounded by parentheses, check the box.
- 4 Click **Preview Changes**. If you are absolutely sure you want to perform the operation, click **Apply Changes**.



- 5 A dialog opens that lets you examine all the changes Exploring Differences on page 269 describes the icons and drop-down menus in this dialog.

Current	Refactored
353 register int *deltazero = delta0;	352 register int j, count;
354 int patlen = altmin;	353 register int *deltazero = delta0;
355 char *t;	354 char *t;
356 char *gotamatch(), *kanji(), *linesav	355 char *gotamatch(), *kanji(), *linesav
357	356
358 nleftover = boyfound = flushflag = 0;	357 nleftover = boyfound = flushflag = 0;
359 nline = 1L;	358 nline = 1L;
360 prevmatch = 0;	359 prevmatch = 0;
361 nmatch = counted = rxcount = 0L;	360 nmatch = counted = rxcount = 0L;
362	361
363 while ((count = read(fd, str + nleftov	362 while ((count = read(fd, str + nleftov
364	363
365 counted += count;	364 counted += count;
366 strend = linesave(str, count);	365 strend = linesave(str, count);
367	366
368 for (k = str + patlen - 1; k < strei	367 for (k = str + altmin - 1; k < strei
369 /*	368 /*

E:\samplecode\fastgrep\egrep.c

OK Cancel

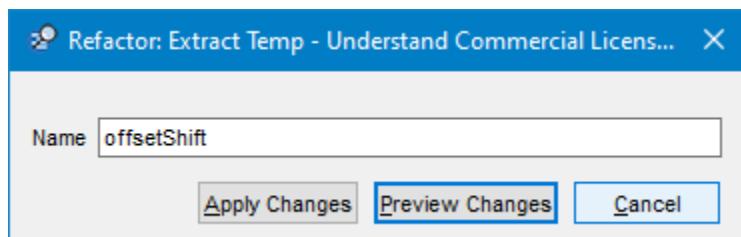
- 6 If you are sure you want to make all the changes, click **OK**.

## Extract Temp

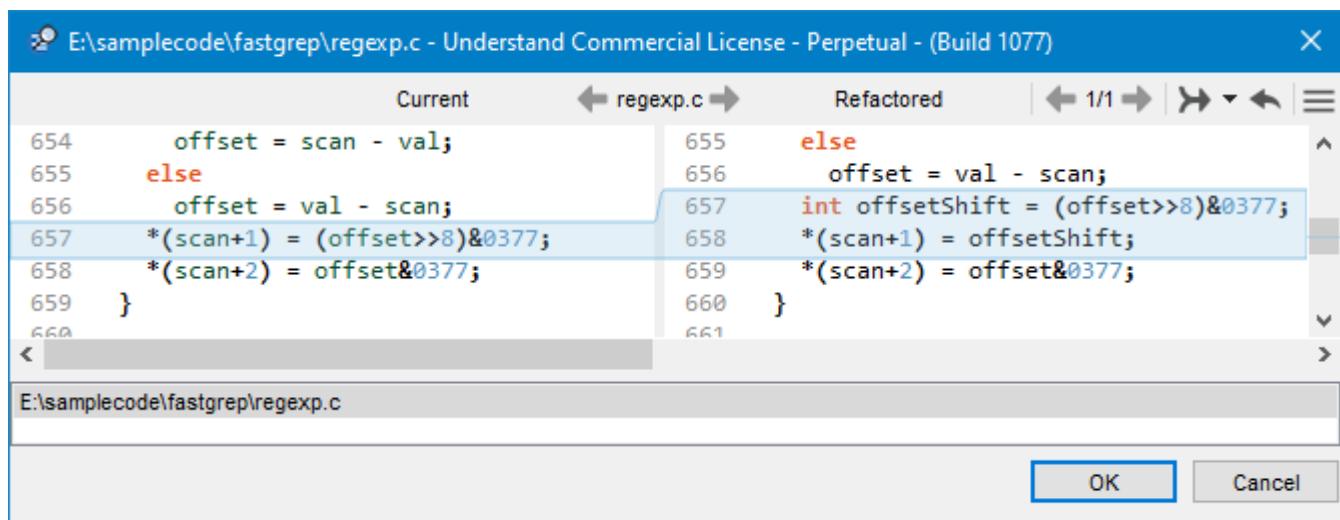
The opposite of inline temp is extract temp. If you have a complicated expression, you may want to assign a part of that expression to a local or temporary variable that can be reused within that function wherever the expression you select is used.

To use the **Refactor > Extract Temp** command, follow these steps:

- 1 Highlight the expression you would like to extract to a local or temporary variable.
- 2 Right-click and choose **Refactor > Extract Temp** from the context menu.
- 3 Type a name for the extracted variable in the dialog and click **Preview Changes**. If you are absolutely sure you want to perform the operation, click **Apply Changes**.



- 4 A dialog opens that lets you examine all the changes Exploring Differences on page 269 describes the icons and drop-down menus in this dialog.



- 5 If you are sure you want to make all the changes, click **OK**.

## Other Editing Features

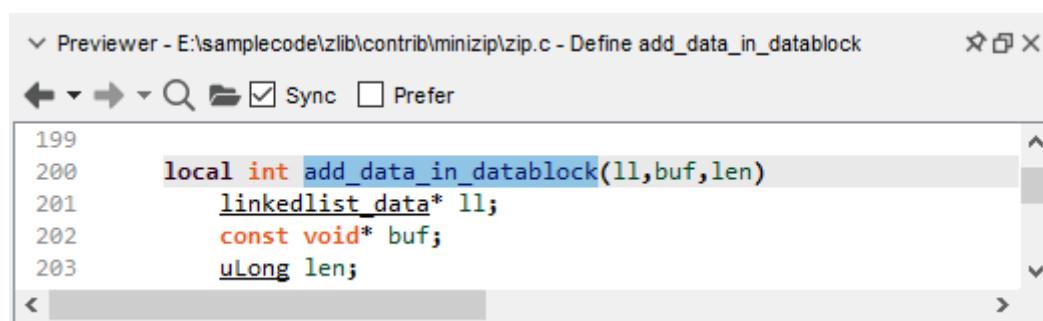
The Source Editor also provides several other options for displaying and editing files:

- Previewer on page 179
- Bracket Matching on page 180
- Folding and Hiding on page 180
- Changing the Source Code Font Size on page 180
- Splitting the Editor Window on page 181
- Changing Case on page 181
- Commenting and Uncommenting on page 182
- Indentation on page 182
- Line Wrapping on page 182
- Insert and Overtype Modes on page 182
- Sorting Lines Alphabetically on page 182
- Keyboard Commands on page 182
- Recording, Playing, and Saving Macros on page 183
- Creating and Opening Files on page 183
- Bookmarking on page 184
- Managing Source Editor Tabs on page 185

### Previewer

The Previewer window is similar to a Source Editor window. To open the Previewer window, choose **View > Previewer**. The differences between the Previewer and the Source Editor are as follows:

- You cannot edit the code in the Previewer window.
- **Sync checkbox;** If this is checked, single-clicking an entity in another view (such as the Entity Filter) opens the location where the entity is defined in the Previewer.
- **Prefer checkbox;** If this is checked, double-clicking an entity in another view opens the location where the entity is defined in the Previewer instead of the Source Editor. (Double-clicking in the Previewer always moves to that line in the Source Editor.)



**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-clicking 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-clicking 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 Keyword**) to move your editing cursor to the match.

**Folding and Hiding**

The triangles 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.

```
167      ▷      if (windowBits < 0) { ...
171      ▽      else {
172          state->wrap = (windowBits >> 4) + 1;
173      ▽      #ifdef GUNZIP
```

Clicking the triangle again or clicking the  green dots in the code opens the folded code.

If you right-click on the code, you can choose **Fold All** to close all the open blocks.

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 right-click and 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.

**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).

## Splitting the Editor Window

You can split a Source Editor window vertically or horizontally so that you can scroll to see more than one location in a file. These split icons are shown in the toolbar:



Click the **Split Vertically** or **Split Horizontally** icon to divide the source editor into two or more separately scrollable panes. You cannot split a Source Editor window in both directions, but you can split it multiple times in the same direction. Click the **Remove Split** icon to remove a split level from the pane in which your cursor is currently placed.

A screenshot of the Source Editor window titled "example.c". The window is split vertically into two panes. The left pane contains lines 515 through 521 of the code. The right pane contains lines 383 through 389. Both panes have scroll bars and are displayed in a monospaced font.

```

example.c
515     int main(argc, argv)
516     int argc;
517     char *argv[];
518     {
519         Byte *compr, *uncompr;
520         uLong comprLen = 10000*sizeof(int); /* don't overflow on
521         uLong uncomprLen = comprLen;

383     void test_sync(compr, comprLen, uncompr, uncomprLen)
384     Byte *compr, *uncompr;
385     uLong comprLen, uncomprLen;
386     {
387         int err;
388         z_stream d_stream; /* decompression stream */
389

```

## 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 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

## 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.

Comments are not analyzed when a project is analyzed.

## Indentation

You can click **Tab Width** in the status bar at the bottom of the window to open a dialog that lets you set the number of columns for each tab stop in this file. This setting is saved separately for each file, and overrides the setting in the Editor category in the Options dialog (see page 105).

CodeChecked: Never | Line: 21 Column: 17 | Tab Width: 8 | RW | Fortran

You can make the indentation of selected code match standard usage by selecting the code, right-clicking, and choosing **Reindent Selection**. Indentation preferences are controlled by the **Editor > Advanced** category in the Options dialog (see page 107).

## 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 display all the code. To do this, right-click in the Source Editor and choose **Soft Wrap**. This wrapping is 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 Overtype 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 Overtype 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 Overtype** 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 > Editor** 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 140) to get to the beginning of the keystrokes for the Source Editor windows.

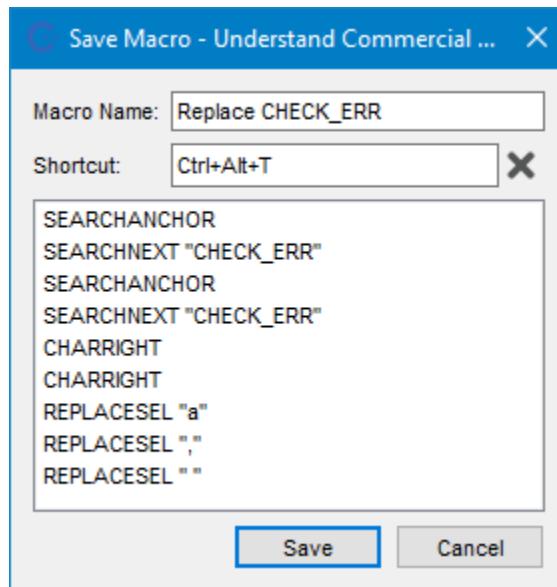
## Recording, Playing, and Saving Macros

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 in the Source Editor.
- 3 Choose **Tools > Editor Macros > Stop Recording** or press Ctrl+Alt+M. (Note that if your cursor is not in the Source Editor at the end of the macro, you will not be able to stop the recording until you move back to the Source Editor.)

To replay the most recently recorded macro, move your cursor to the desired start location and choose **Tools > Editor Macros > Replay Macro** or press Ctrl+M.

You can save the most recently recorded macro by choosing **Tools > Editor Macros > Save Macro** or pressing Ctrl+Shift+M. You will be asked to type a name for the macro. You can also move to the Shortcut field and press the key combination you want to use to trigger this macro.



You can rename and delete saved macros in the Understand Options dialog by choosing **Tools > Editor Macros > Configure Macros**. See page 110 for details.

## 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(s)**. For example, the companion file of encrypt.c is encrypt.h.

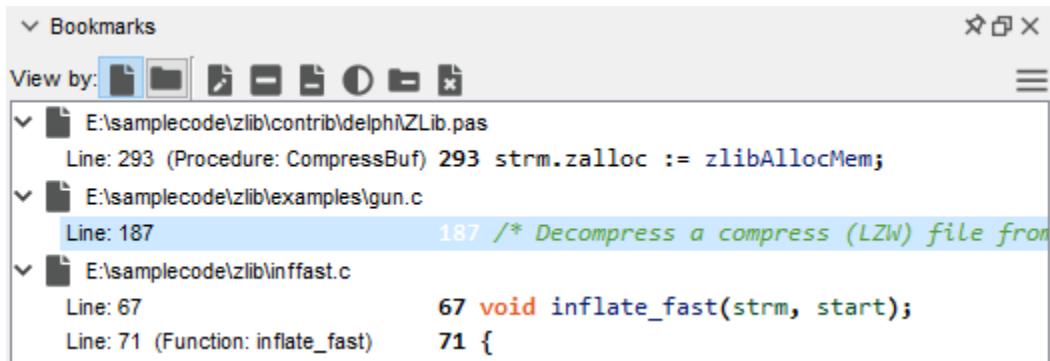
## Bookmarking

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.

67      ► `void inflate_fast(strm, start);`

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.

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.



If you point to bookmarked code in the Bookmarks area, the 5 lines of code surrounding the bookmarked line are shown in the hover text.

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 a line of code inside a function.

Bookmarks and Favorites (page 142) are stored locally for the user. If you want to share code locations with others who use the same project, see *Annotations* on page 186.

The toolbar for this area lets you manage your bookmarks in the following ways:

Use the **View by** icons to switch between a file-based and a category-based view. The file-based 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. To create a category, type a 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 bookmark and click this icon to mark it as a temporary bookmark to be deleted 24 hours after marking it as temporary.

 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.

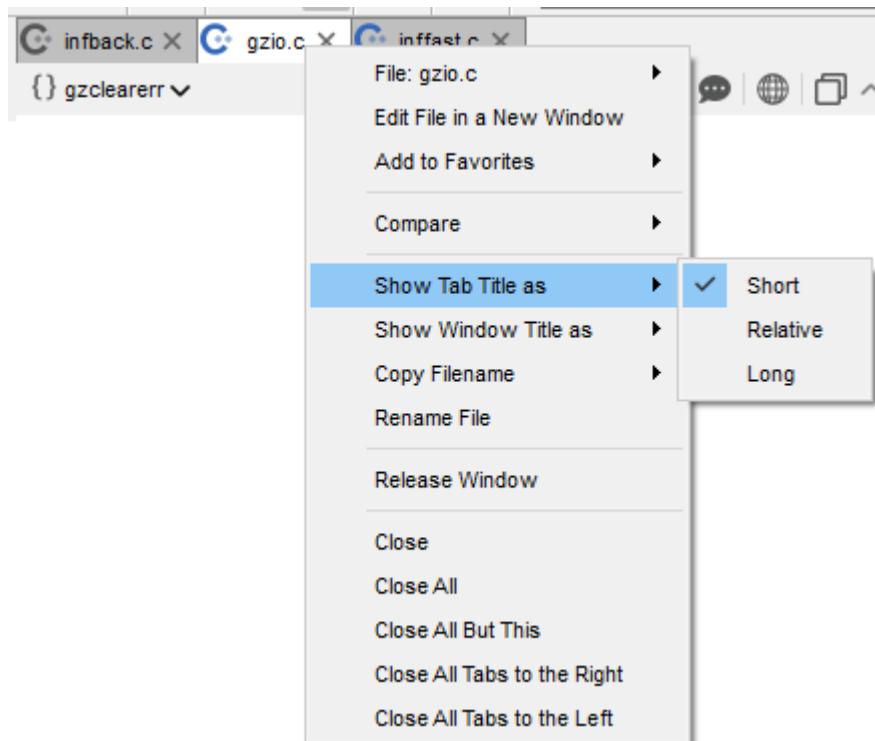
 Click this icon to delete all your bookmarks.

The hamburger menu  in the upper-right corner of the Bookmarks area provides the following commands:

- **Copy to Clipboard on Double Click:** Double-clicking on a line in the Bookmarks list jumps to that location in the code. If you enable this option, double-clicking both jumps to the location in the code and copies that line of code to your clipboard.
- **Show Original Indentation:** Enable this option to display the code line with indentation matching the source code indentation.

## 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.

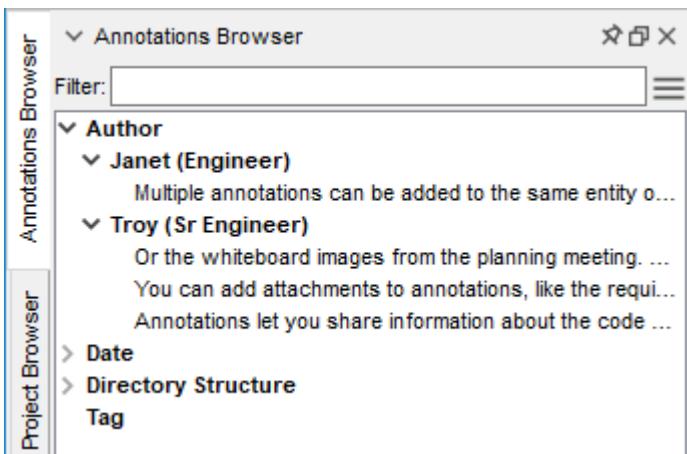
If you choose **Release Window**, the tabbed area changes to a separate window that can be moved around your screen. Click the  icon in the upper-right corner of a released window and choose **Capture Window** to return the window to a tab within the *Understand* window.

## Annotations

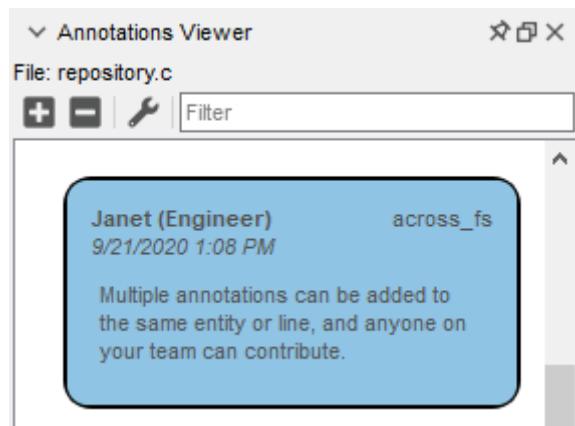
Annotations are an easy and convenient way for Understand users to add documentation without making changes to the source code directly, yet still have the comment where you can see it quickly, right in context with the code. Annotations can apply to lines of code, entities, or files. You can attach files to annotations and share them with the whole team.

You can view annotations inline with the code or in the Annotations Browser or the Annotations Viewer. To open the Annotations Browser, choose **Annotations > Browse All Annotations** from the menus. To open the Annotations Viewer, choose **Annotations > Annotation Viewer** from the menus.

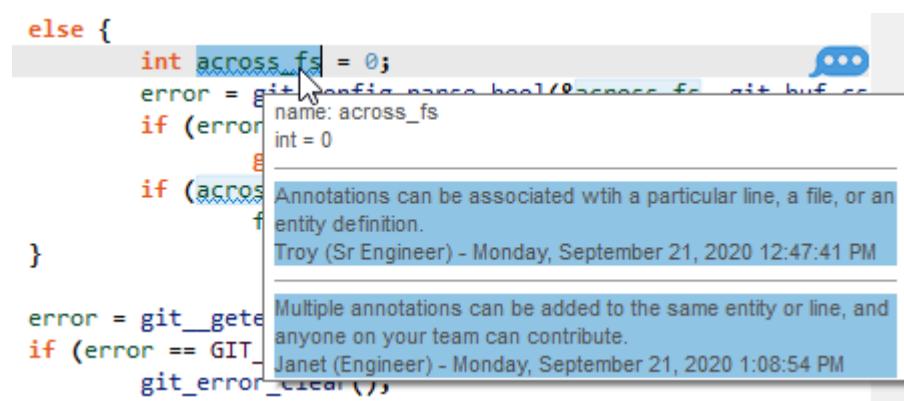
### Annotations Browser



### Annotations Viewer



Annotations can also be seen in hover text wherever the annotated entity is used, including in graphs and in the Information Browser.



Annotations can be “tagged” with a hashtag, such as #review or #refactored. Such tagging is useful for organizing to do lists or assigning questions to people.

## Viewing Annotations

By default, annotations are indicated within a Source Editor window by an  icon with three dots at the right edge of the window. If the Annotations Viewer is not open and you point at one of the icons, the annotation is shown in a hover window. If you click the icon or the hover window, the Annotations Viewer opens with that annotation selected. If the Annotations Viewer is open, clicking an indicator icon selects that annotation in the Annotations Viewer. If an entity is annotated, pointing to a use of that entity shows the annotation in the hover text.

You can change the default behavior using settings in the **Annotations** category in the **Tools > Options** dialog (see page 118) as follows:

- Choosing “Pop-up Window” for the **Open Annotations in** field disables opening the Annotations Viewer.
- Choosing “Inline” for the **Editor Display** field hides the indicators and instead shows annotations inline with the source code.
- Choosing “None” for the **Editor Display** field hides both annotations indicators and inline annotations.
- Unchecking “Show Annotations in Entity Hover Text” hides annotations when pointing to an annotated entity.

You can also use the Annotations category of the Options dialog to set the name shown for your annotations, to control the colors for annotation text and backgrounds, and to hide or show annotations when printing.

## Sharing Annotations

Annotations are designed to be shared with the rest of your team. The annotations are stored within the project folder (named `project_name.und`). See *Project Storage* on page 32 for more about how projects are stored and shared.

Within the `project_name.und` folder, a separate JSON file stores the annotations created by each user of the project. These files are named `ann_username.json`. A `media.json` file lists any files that have been attached to annotations, and the files are stored in the `media` subfolder of the project folder.

You can export annotations by opening the Annotations Browser and choosing **Export Annotations to CSV** from the  hamburger menu.

## Searching for Annotations

To open the Annotations Browser, choose **Annotations > Browse All Annotations** from the menus. This browser sorts annotations by author, date created, file directory structure, and any tags you have created. You can hide any of these groupings by clicking the  icon for the hamburger menu.

To search for annotations, type in the **Filter** field. This searches the text of the annotations. By default, the search is case insensitive and does not allow wildcards or regular expressions. You can change these defaults using the hamburger menu.

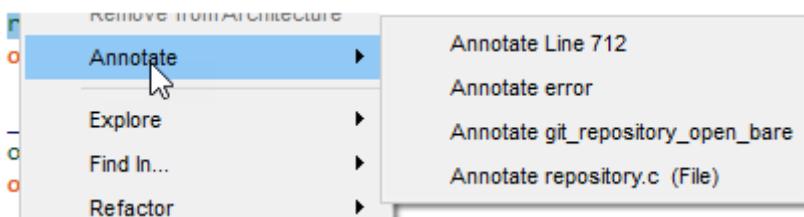
You can also search for annotations in the Annotations Viewer by typing in its **Filter** field. Note that the Annotations Viewer shows only annotations in the currently viewed source file, while the Annotations Browser shows all annotations in the project.

The Information Browser also lists any annotations for the selected entity.

## Adding Annotations

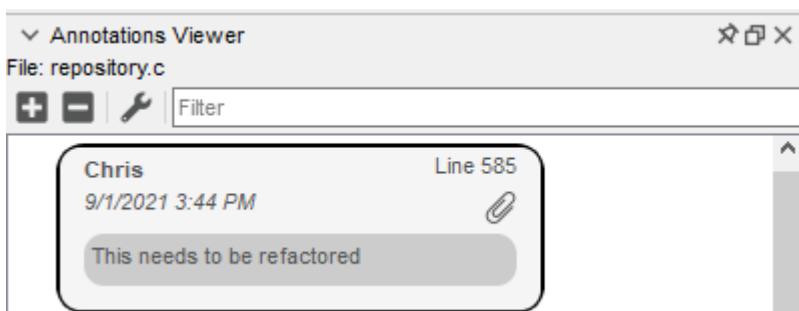
You can add an annotation in the following ways:

- Right-click on an entity or filename almost anywhere in *Understand* and choose **Annotate** from the right-click menu. For example, you can select an entity in source code, in the Information Browser, or in the Entity Filter. Depending on the type of entity selected, you can choose to annotate that entity (in all places it occurs), the containing entity (such as a function or class), the source file line number, or the source file. (Line numbers cannot be annotated in a file that has not yet been saved and analyzed as part of the project. Annotations to a line number are updated if possible when the line number changes.)



- Select an entity and choose from the **Annotations > Annotate** menu.
- Select a location or entity in a source file and click the **+** plus icon in the Annotations Viewer. If you select an existing annotation in the Annotations Viewer and click the **+** plus icon, you can reply to the selected annotation.

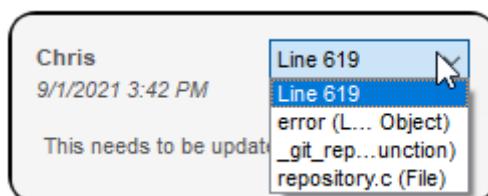
Whenever you create an annotation, the Annotations Viewer opens and a new annotation is added. Type text for your annotation, which is saved automatically when you click outside the annotation.



## Editing Annotations

To edit the text of an annotation you created, just click in the text field. The timestamp in the annotation is updated when you edit it. You cannot edit annotations created by other users, though you can delete them (page 190).

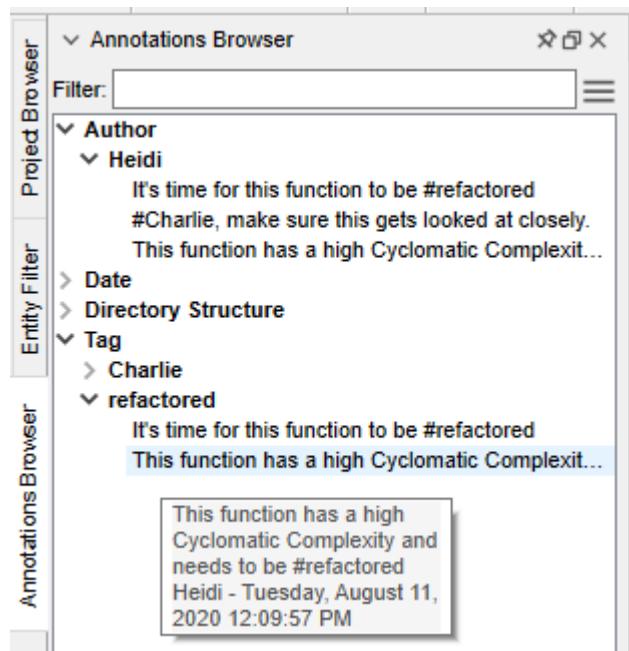
You can change the context for annotations you created. Click on the line or entity in the upper-right corner of an annotation and select the new context.



## Tagging Annotations

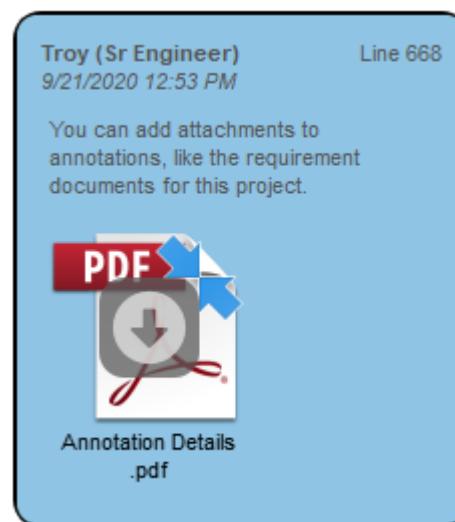
Annotations let you add tags that provide an easy way to create to-do lists, prioritize work, and assign tasks. To add a tag, simply type # and a keyword (as you would in social media applications). For example, you might use #Project7 or #Refactor.

To type a literal # sign without making it a tag, type ##.



## Attaching Files

To attach a file to an annotation, click the paper clip icon in an annotation you have created and browse for the file to attach. You can attach multiple files to the same annotation. See *Sharing Annotations* on page 187 for details on how these files are stored and shared.



## Deleting Annotations

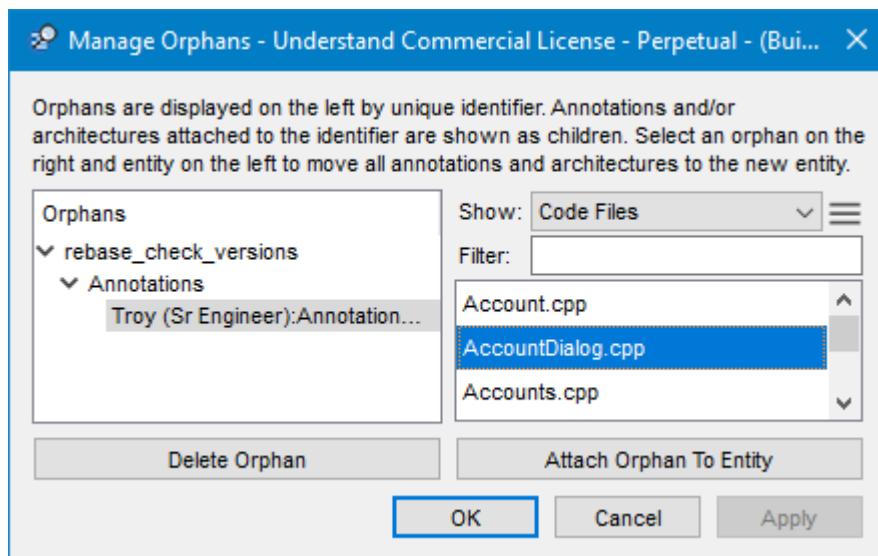
To delete an annotation, select it in the Annotations Viewer and click the  minus icon.

**Note:** Be careful not to select an annotation and press the Delete key, because this will delete the highlighted entity in the Source Editor instead of the annotation.

## Managing Orphan Annotations

If you delete the code to which an annotation was attached, the annotation is still part of the project, but it is an “orphan” without anything to be attached to. You can find such annotations and attach them to other files or entities:

- 1 Save your source files and choose **Project > Analyze Changed Files**.
- 2 Choose **Annotations > Manage Orphans**.
- 3 In the Manage Orphans dialog, expand the list of orphan annotations and select an orphan. You can enlarge the dialog to read more of the annotation text.
- 4 Use the entity filter on the right to find and select an entity.
- 5 Click **Attach Orphan to Entity**.
- 6 You may continue attaching other orphan annotations to entities.
- 7 You may click **Delete Orphan** to remove an annotation from the project completely.

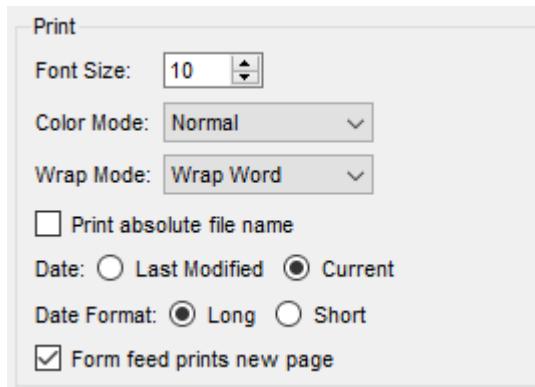


**Note:** Architecture nodes can also be orphaned. This same dialog is used to manage orphaned architecture nodes.

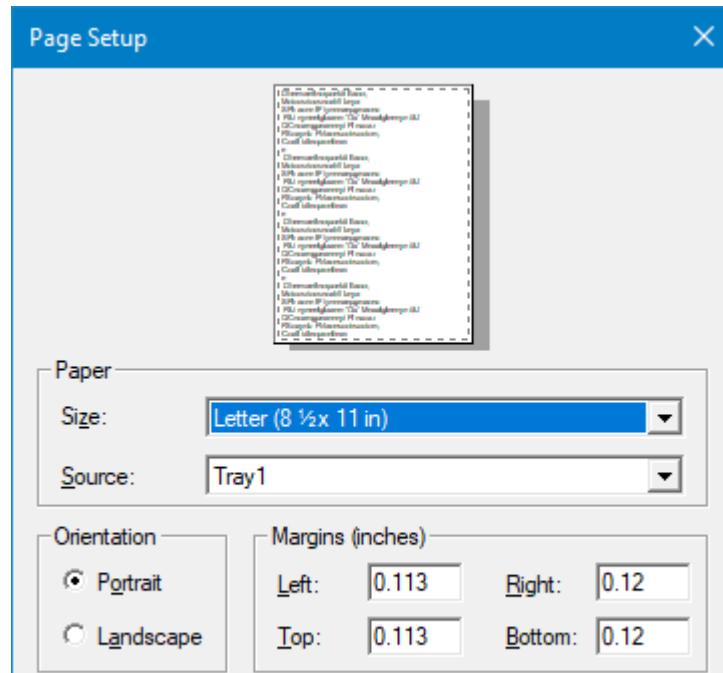
## Printing Source Views

The menu option **File > Print** opens the standard print dialog for your operating system so you can print the currently viewed source file.

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. Options to control how code is printed are in the Print area. See *Editor > Advanced Category* on page 107 for details about these fields.



To change the print output without changing the online display, choose **File > Page Setup** from the menus. This dialog offers printing options similar to the following; the options may differ depending on your operating system:



Any annotations in a file are printed along with the source code. See page 186.

---

## Chapter 8     Architecting Your Codebase

*Understand* now provides a completely revised and expanded interface for creating and using architectures, which are virtual hierarchies for organizing and analyzing your code. Our users create architectures for many purposes, including:

- Owner-based architectures, which show the division of code amongst its primary owners.
- Metrics-based architectures, such as architectures categorized by code complexity or file or function size.
- Date-based architectures. For example, large refactoring projects can use this to query which files have been updated and which still need work.
- Functional unit based architectures used to view dependencies between different units. For example, what dependencies does the core code have on external libraries?

This chapter contains the following sections:

---

Section	Page
About Architectures	193
Browsing Architectures	194
Creating and Editing Custom Architectures	197
Sharing Architectures	201
Viewing Architecture Graphs	201
Checking Dependencies	203
Viewing Architecture Metrics	204

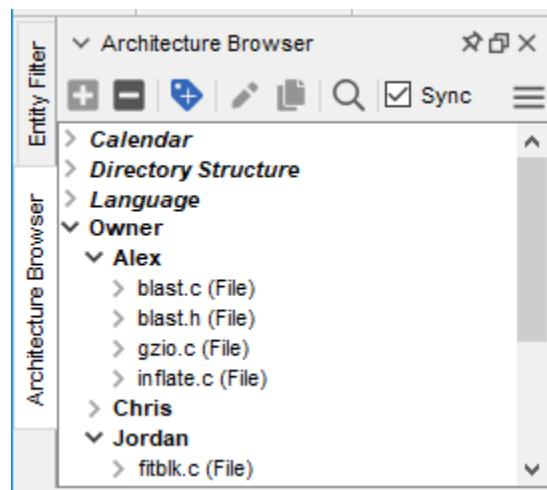
---

## About Architectures

An *architecture* in *Understand* is an abstract hierarchy layered onto a body of source code. An architecture creates a hierarchy of source code units (entities). Architectures allow you to name and organize parts of a software project or ways of looking at software hierarchically.

*Understand* provides built-in architectures that organize source code files by date last modified, directory structure, and language (for multi-language projects). You can use these provided architectures and/or create your own.

For example, a staff-based architecture could contain 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. You can reassign responsibilities by dragging folders, files, or entities within the architecture. The “Hide by” option in the Project Browser lets you see whether you have assigned all parts of the project to a node in the architecture. Dependencies and interactions between nodes can then be graphed or listed using the architecture.



Architectures organize any entity types analyzed by *Understand*. They aren't limited to organizing files. For example, the “Possible Security Violations” architecture shown here contains various entity types to be examined for possible security issues.



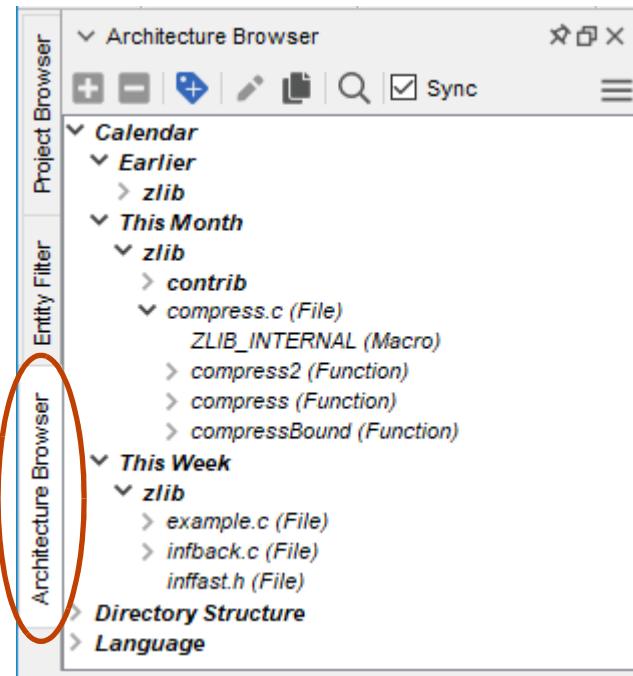
Architectures are saved as part of a project and so can be used collaboratively with others who work with the same project.

Architectures need not reference every source entity in the project; that is, they can define subsets of project entities. Also, architectures can reference 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, 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 project change and it can be used to reconstitute the filter at a later date.

## Browsing Architectures

To open the Architecture Browser, choose **Architectures > Browse Architectures** from the main menu bar.



You see an expandable list of the architectures currently defined for your project.

This Architectures area is similar to the Entity Filter or Project Browser 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).

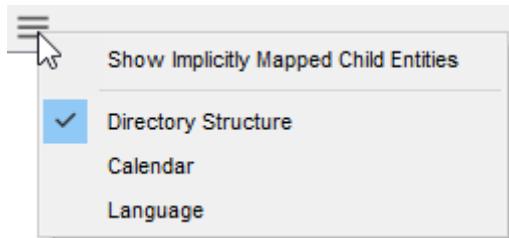
To explore architectures, click the > arrows to expand the hierarchy. Entities, such as files, functions, and variables are shown in the hierarchies. Within a node, children are sorted alphabetically.

You can choose whether to list entities within each file by clicking the hamburger menu icon and toggling the **Show Implicitly Mapped Child Entities** option.

You can use the  Search icon in the Architecture Browser toolbar to find open the Find in Files dialog (page 150) to search within architectures. See page 197 to use the other toolbar icons to create and edit architectures.

## Enabling Built-In Architectures

*Understand* provides some “auto-architectures” that are predefined. By default, only the Directory Structure architecture is shown in the Architecture Browser. You can enable the other built-in architectures by clicking the hamburger menu icon and choosing an architecture:



- **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.)
- **Visual Studio Projects:** This architecture is provided only if the project is configured to contain a Visual Studio project. (Existing projects must be reanalyzed for this architecture to be created.)

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.

## Context Menus for Architectures

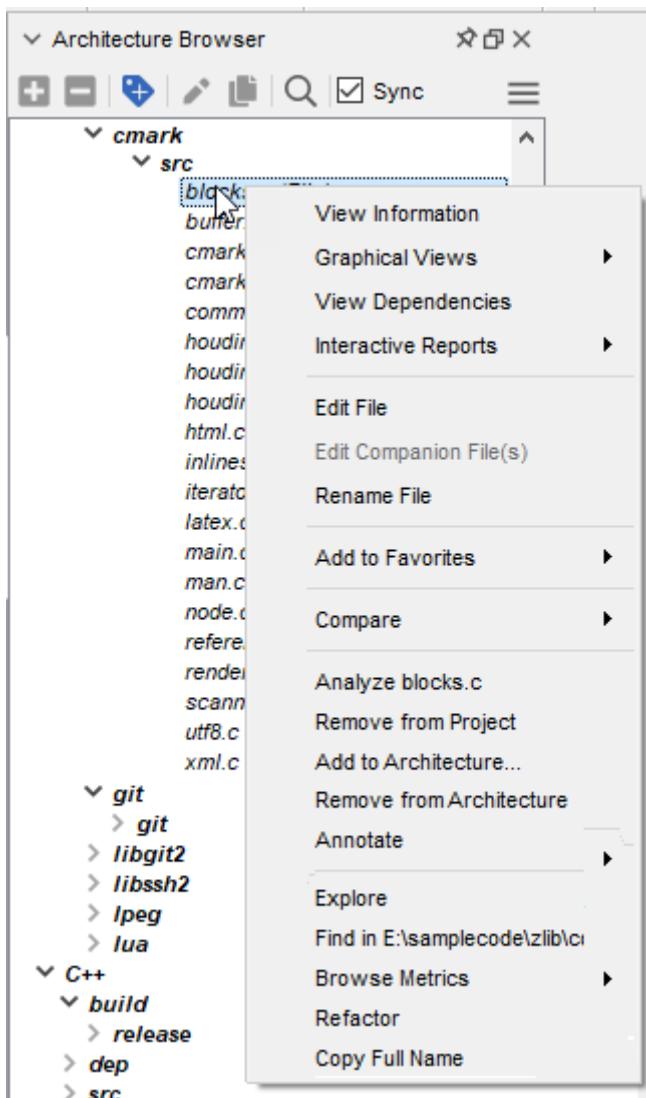
The context menu for an architecture node (such as a filesystem directory or “This Quarter” contains some extra items not available in other context menus:

- **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 202.
- **Graphical Views > Internal Dependencies:** Shows the dependencies between architecture nodes. See page 201.
- **Add to Architecture:** Adds the contents of a selected node, file, or entity to another architecture node. You cannot edit the auto-architectures provided with *Understand*. See page 197.

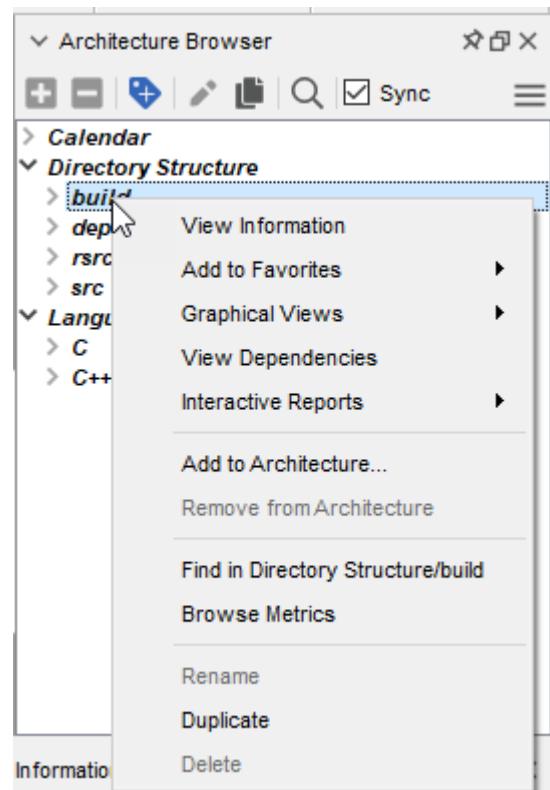
- **Remove from Architecture:** Removes the selected node or file from the architecture or node you select. You cannot remove files from the auto-architectures provided with *Understand*. See page 198.
- **Browse Metrics:** Shows metrics for entities within the selected node (but not including entities in sub-nodes lower in the hierarchy). See page 208.
- **Rename:** Lets you rename an architecture or node if it is one you created. You cannot rename the auto-architectures provided with *Understand*. See page 197.
- **Duplicate:** Opens a Duplicate Architecture window that lets you type a name for a duplicate copy of the selected architecture. See page 197.
- **Delete:** Delete the selected architecture node. See page 197.

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.

#### Right-click on file in Architecture



#### Right-click on Architecture node



## Creating and Editing Custom Architectures

Creating custom architectures is easy to do in the Architecture Browser. (In previous versions of *Understand* it was more complicated.) Changes you make to custom architectures are automatically saved. You can also use the Architecture Designer to visually edit a custom architecture; see page 199.

### Creating Architecture Nodes

Use the toolbar icons in the Architecture Browser as follows to create and modify custom architectures:

-  **Add Architecture or Node:** See page 197.
-  **Delete Architecture Node:** Custom architectures only. See page 197.
-  **Tag Entities:** See page 198.
-  **Edit architecture:** Rename the selected architecture or node. Custom architectures only. See page 197.
-  **Duplicate architecture:** See below.

You cannot modify the built-in architectures (Directory Structure, Calendar, and Language), so the Add, Delete, and Edit icons are grayed out when you select a node in these architectures.

To create a new architecture, deselect any nodes or files in the Architecture Browser and click the  **Add** toolbar icon. Type a name for the architecture that is added.

To create a new node within an architecture, select the parent node and click the  **Add** toolbar icon. Type a name for the new node.

Another way to create custom architectures is to select an existing architecture or node and click the  **Duplicate** toolbar icon. Type a different name for the duplicated architecture in the dialog. The duplicate is created as a new architecture, but you can drag it to a location within a custom architecture. You can also drag nodes of a built-in architecture to a custom architecture to create a duplicate. All the children of a node are duplicated along with the node.

### Editing Architecture Nodes

You cannot edit the built-in architectures provided with *Understand*. However, you can edit custom architectures by moving, renaming, and deleting nodes and by adding files and entities to nodes.

**Moving Nodes:** You can drag custom architectures or nodes of custom architectures within the custom architectures as needed. Any children of a node are moved along with the node. If you drag a built-in architecture or a node of a built-in architecture to a custom architecture, the node and its children are copied instead of moved. This is because the built-in architectures cannot be modified.

**Renaming Nodes:** To rename a custom architecture or custom architecture node, right-click and choose **Rename** from the context menu. Type a new name in the Architecture Browser list

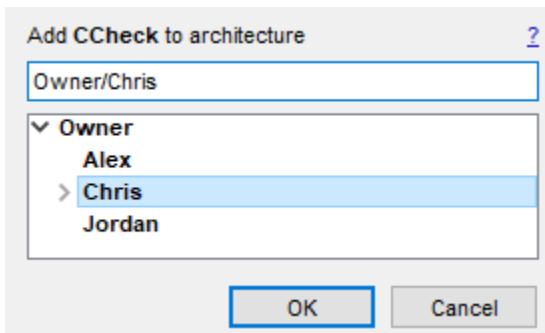
**Deleting Nodes:** To delete a custom architecture or custom architecture node, right-click and choose **Delete** from the context menu.

## Adding Files and Entities to Nodes

Use any of the following methods to add files and entities that you select anywhere within *Understand* to custom architectures:

- Click the  **Tag Entities** toolbar icon in the Architecture Browser.
- Choose the **Architectures > Add <entity name> to Architecture** menu command.
- Right-click and choose **Add to Architecture** from the context menu.
- Drag and drop a file or entity from anywhere in *Understand* to the Architecture Browser.

These methods (except drag-and-drop) ask you to select a custom architecture or node to contain the selected files or entities. You can type an architecture name as a path separated by slash “/” characters. For example, “Owner/Chris”.



To remove files or entities from an architecture, right-click on a file or entity and choose **Remove from Architecture**. If the file or entity is referenced in multiple places in the custom architectures, you will be asked which architecture path to remove it from.

## Complete Coverage for Architectures

The built-in architectures provide complete coverage to organize the files in a project by date last modified, directory structure, and language. Custom architectures do not need to contain all files in a project. Often you only want parts of a project in an architecture.

If you do want to make sure that a custom architecture contains all files in the project, follow these steps to use the Hide By Architecture feature:

- 1 Open the Project Browser and the Architecture Browser.
- 2 Undock the Project Browser and move it next to the Architecture Browser so you can see both areas.
- 3 In the Project Browser, click the  hamburger menu icon and choose **Hide By** and the name of your custom architecture. This hides any files or directories that are already in your architecture.
- 4 Drag files and directories from the Project Browser to the desired location in your architecture. Create new nodes as needed by clicking the  **Add** toolbar icon.
- 5 When the Project Browser is empty, all the files in your project are referenced in your custom architecture.
- 6 In the Project Browser, click the  hamburger menu icon and choose **Hide By > None** to display all the files again.

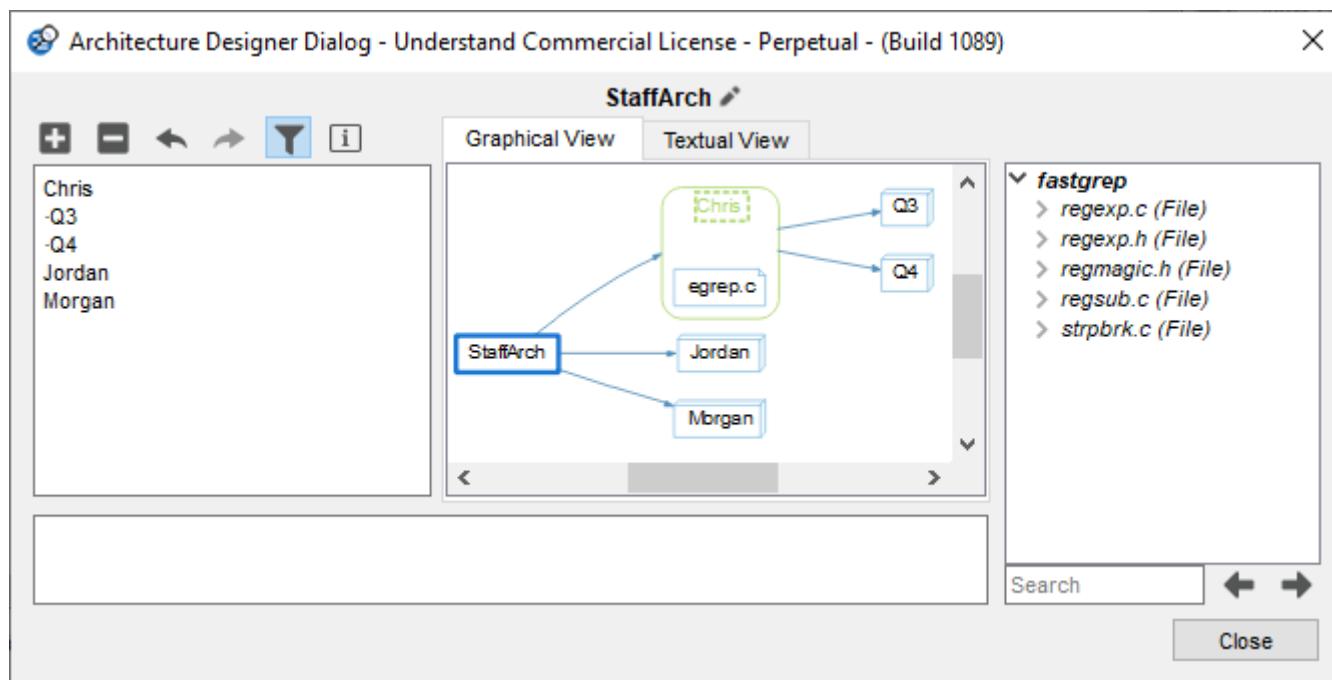
## Managing Orphan Architecture Nodes

Architecture nodes can become “orphans” if their parent node is deleted. To delete or attach orphaned nodes, choose **Architectures > Manage Orphans** from the menus. See page 190 for details.

## Using the Architecture Designer

You can also use the Architecture Designer to edit a custom architecture by creating an outline and dragging entities to nodes in the Graphical View or the Textual View. The Architecture Designer makes it easier to visualize your architectures while creating them. The Architecture Designer facilitates a top-down method for creating an architecture, while the Architecture Browser requires a more bottom-up approach.

To open the Architecture Designer, choose **Architectures > Design Architectures > *architecture\_name*** from the menus. Or, in the Architecture Browser, right-click on a custom architecture (not the built-in architectures) and choose **Edit Architecture in Designer**.



Changes made in the Architecture Designer are saved immediately. There are undo and redo icons, but closing the Designer removes the ability to undo changes made in the Designer.

The Architecture Designer contains the following panes and controls:

- **Architecture name:** Click the pencil icon next to the name of the architecture if you want to modify the name of the architecture.
- **Outlining pane:** Build the organization of the architecture by outlining it in this pane. Indent with spaces or tabs to nest levels within the architecture. Your outline should include nodes for organizing the architecture, but not the entities that will be

organized. The outline becomes the nodes of an architecture in both the Graphical View and the Textual View tabs. Click the  information icon to see an example functional decomposition outline.

- **Graphical View tab:** Edit the architecture using this diagram by dragging entities to nodes. You can drag entities within the graph or from the Directory Structure pane on the right. Expand or collapse nodes that contain entities (shown as a 3D box instead of a rectangle) by double-clicking. Use the scroll wheel to zoom in or out.
- **Textual View tab:** Edit the architecture using this list by dragging entities and nodes as needed. You can drag entities within this view or from the Directory Structure pane on the right. Expand or collapse nodes by clicking the arrows.
- **Directory Structure pane:** The built-in Directory Structure architecture is shown in this pane so that you can expand it as needed and drag entities to the Graphical View or Textual View tabs. You can use the **Search** field below this pane to search for entities by name.
- **Source view:** The source code that declares the selected entity is shown and highlighted in this box.

The general procedure for creating or editing an architecture using the Architecture Designer is as follows:

- 1 Choose **Architectures > Design Architectures > New Architecture** from the menus to create a new custom architecture. You can also open an existing custom architecture using this menu command.
- 2 Click the  pencil icon and type a name for your new architecture. Press Enter to save the name.
- 3 In the left panel, type an outline for your architecture. Include the groups to organize your architecture, but none of the entities. Use spaces or tabs to nest levels. For example, if you are creating an architecture that shows the responsibilities for parts of your codebase, your outline could include a list of engineering groups or engineers. If you are creating an architecture that organizes code to be modified in different releases, your outline could include a list of planned releases.

Notice that as you add to the outline, the structure for the outline is shown in the Graphical View tab.

- 4 Drag directories or files from the Directory Structure pane on the right to the correct nodes in the Graphical View tab.
- 5 If you want all of your code to be assigned to a node in the architecture, toggle on the  Filter icon above the left pane. Directories and files that have already been assigned to a node will be hidden.
- 6 Click **Close** when you have finished editing your architecture.

## Sharing Architectures

You can share architectures with other developers by exporting an architecture to an XML file and sharing the file.

To export an architecture, open the Architecture Browser. Select the architecture you want to export. From the hamburger menu, select the **Export Selected Architecture** option.

To import an architecture, open the Architecture Browser. From the hamburger menu, select the **Import Architecture** option.

## Viewing Architecture Graphs

You can generate graphs that show the hierarchy of an architecture. See Chapter 10, Using Graphical Views for details about all the types of graphs provided by *Understand*.

To view 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. A number of graph categories are available; see page 217 for more about opening graphs. Choose a graph category. The **Dependencies** and **Graph Architecture** categories are specific to architectures, and are described in the sections that follow.

When you have selected an architecture node, the same list of graphical views is available by choosing **Graphs > Graphs for <node>** from the menus.

Architecture graphs have the same toolbar as other types of graphical views. See page 222 for details about using the icons in the graphical view toolbar.

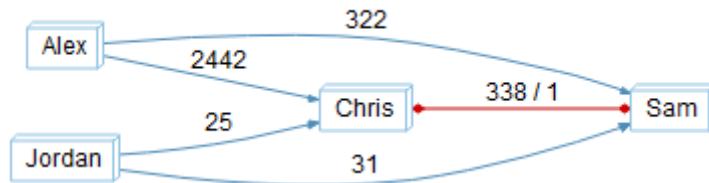
Architecture graphs can be quite large. Remember that you can use Ctrl+F to search within a graph.

You can save these graphs as PNG, JPEG, SVG, Visio XML, and DOT files.

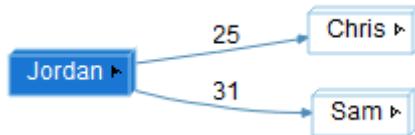
You can also use the Dependency Browser to see dependencies within an architecture. See page 135.

### Dependency Graphs

To open a graph that shows the internal dependencies within an entire architecture, choose an architecture from the **Graphs > Dependency Graphs** menu. For example, if you have a custom architecture that organizes source code by its owner, the graph might look like this. Blue lines indicate one-way dependencies and red lines indicate mutual dependencies. The numbers show the number of dependencies for each relationship. You can right-click on a number to see a list of the dependencies.



To see a graph of the dependencies between an architecture node and other nodes in the same architecture, right-click on a node in the Architecture Browser and select **Graphical Views > Dependencies**. By default, the **Butterfly** variant of the Dependencies graph is shown, which includes both nodes that depend on this node and the nodes that this node depends on. In this example, the code managed by Jordan depends on code managed by both Chris and Sam, but not Alex.



Use the **Variant** drop-down list to change the graph to show **Internal** dependencies within the selected node only, nodes **Depended On By** this node, or nodes that this node **Depends On**.

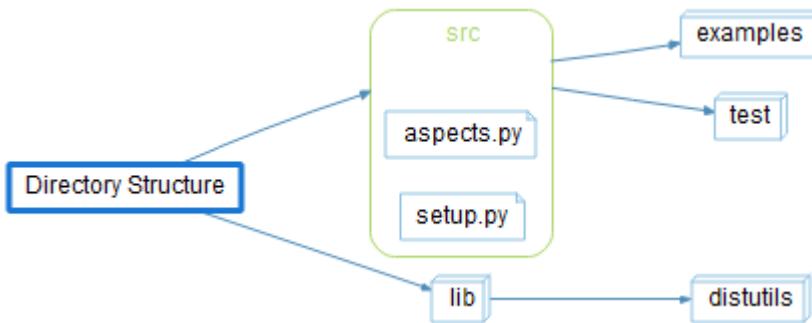
You can expand architecture nodes as needed to see dependencies between specific source files within different architecture nodes.

- Note:** Dependency graphs are also available for classes and packages, typically as a variant of the Butterfly graph category. Select **Dependencies** in the Variant drop-down list (see page 220).

---

## Architecture Structure Views

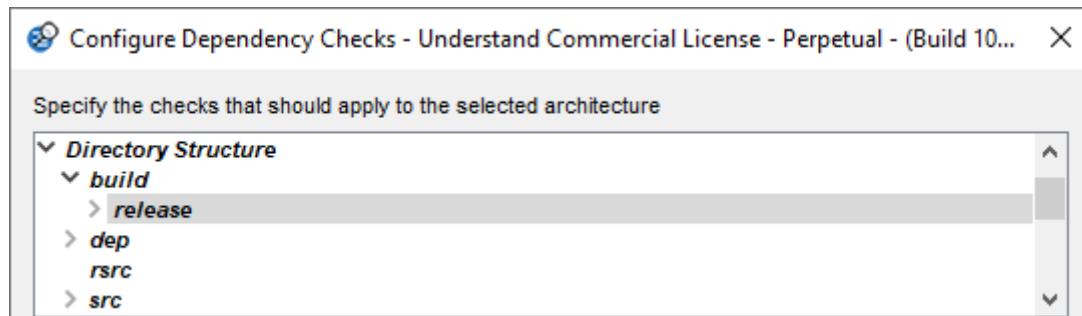
The **Graph Architecture** view is different from the Architecture Dependency graphs. It shows the structure of the architecture, rather than dependencies between entities in the architecture. Open the Graph Architecture view the same way you would open other graphical views (see page 222).



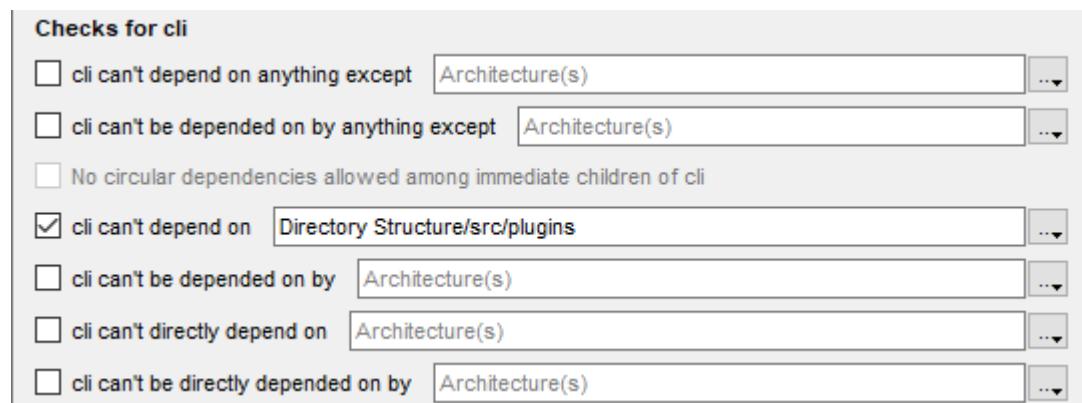
## Checking Dependencies

You may want to verify that certain architecture nodes do or do not depend on other nodes or make circular references. To configure such checks, follow these steps:

- 1 Choose **Checks > Configure Dependency Checks** from the menus.
- 2 Select a node within an architecture from the list at the top of the dialog. (You will be able to perform checks on multiple nodes by returning to this step to add multiple checks to the configuration.)



- 3 Check boxes next to checks you want to perform and select the appropriate architecture node for those checks. For example, you might specify that the "src/cli" node can't depend on the "src/plugins" node.



- 4 When you add a dependency check, it is listed in the **Configured Check** list. A check you add may result in more than one configured check being listed. For example, specifying that the "src/cli" node can't depend on the "src/plugins" node causes both of the checks to be added to make sure that "src/cli" does not depend on "src/plugins" and that "src/plugins" is not depended on by "src/cli".

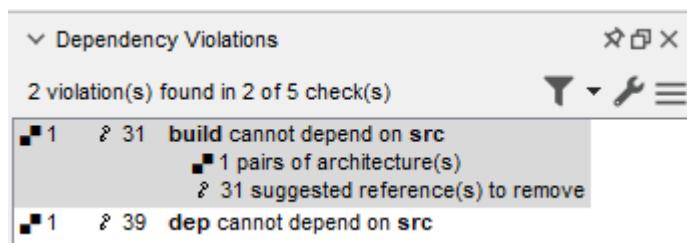
Configured Checks	
Architecture	Check
Directory Structure/src/cli	Cannot depend on Directory Structure/src/plugins
Directory Structure/src/plugins	Cannot be depended on by Directory Structure/src/cli

- 5 You can double-click on a configured check in the list to see the checks configured for that architecture node.
- 6 To configure more checks, select another architecture node at the top of the dialog, and follow the same steps to add checks for that node.
- 7 When you have finished adding checks, click **Save & Run**.

After you have configured checks, you can choose **Checks > Run Dependency Checks** from the menus to re-run the checks.

When you run dependency checks, any violations are shown as follows:

- The Dependency Violations area lists the violations found.



- A graph opens to show the relationships that violate the check selected in the Dependency Violations area. You can expand the nodes and entities to see the locations of the violations. Selecting lines in the graph causes the specific references to be shown in the Dependency Browser. Highlighted lines in the graph are references we recommend removing to break that dependency.
- The Dependency Browser opens to show the list of relationships that violate the check selected in the Dependency Violations area.

The **Filter** icon in the toolbar of the Dependency Violations area allows you to limit the violation list to show only violations in Uncommitted Changes, a specific architecture, or a custom filter based on when the files changed or their location in an architecture. See page 245.

The **Configure** icon reopens the Configure Dependency Checks dialog.

The hamburger menu lets you choose how to **Expand** and **Sort** the list of violations. You can uncheck the **Show Graph View** or **Show Dependency Browser** options to prevent these areas from being opened when you run dependency checks. Choose **Export csv** to send the list of violations to a comma-separated values file.

---

## Viewing Architecture Metrics

You can view basic metrics about an architecture, architecture node, or file in the Information Browser and can view and export detailed metrics in the Metrics Browser. Metrics information can be exported as either a comma-separated list for use in spreadsheets or an HTML-based report.

For more about metrics, see Chapter 9, Using Metrics.

---

## Chapter 9     Using Metrics

This chapter describes how to create and view metrics and the types of metrics available.

This chapter contains the following sections:

---

<b>Section</b>	<b>Page</b>
About Metrics	206
Project Overview	207
Metrics Browser	208
Metric Definitions	209
Exporting Metrics	210
Metrics in the Information Browser	212
Metrics Treemap	213

---

## About Metrics

*Understand* provides a number of ways to gather metrics information:

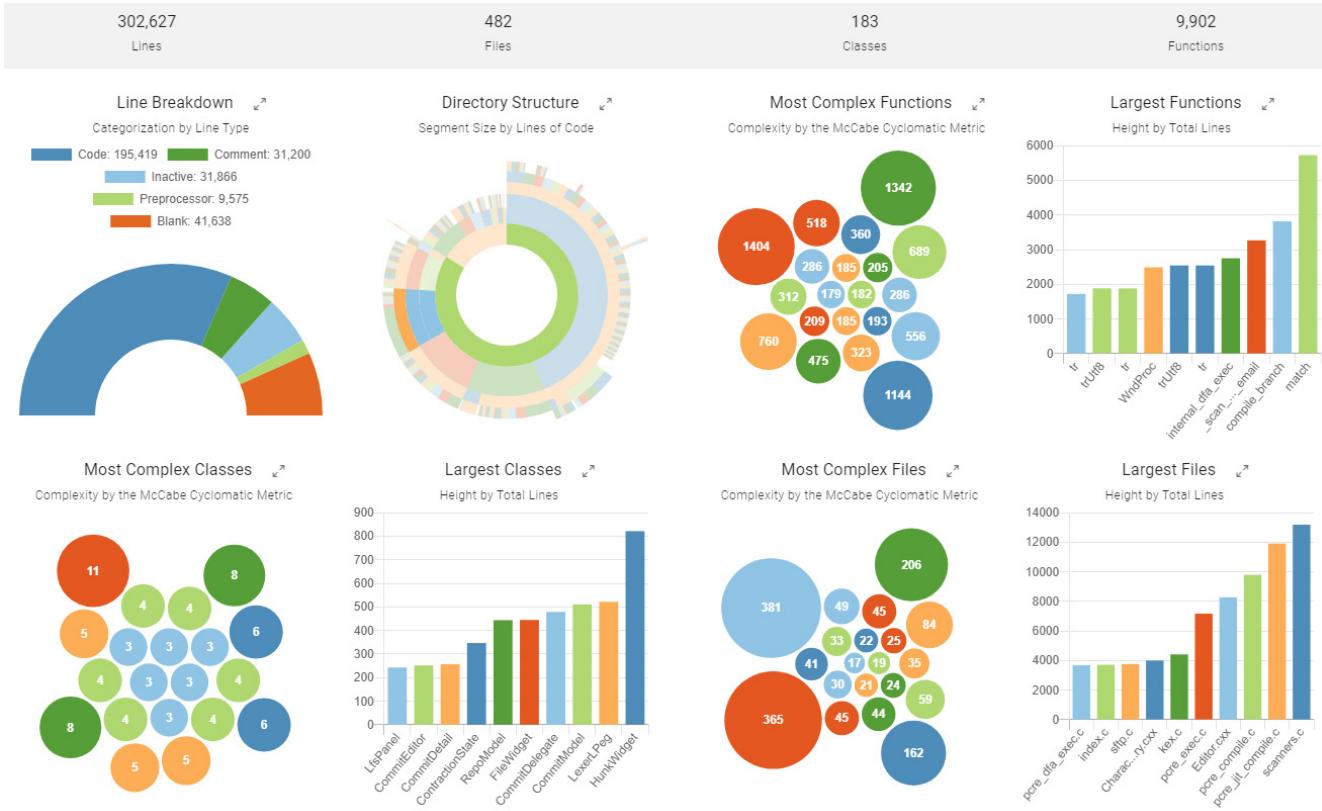
- **Project Overview:** Choose **Project > Overview** to see graphs of metrics for the overall project, including the most complex and largest functions, classes, and files. See page 207.
- **Metrics Browser:** You can choose **Metrics > Browse Metrics** from the menus to see a browser that lets you choose any architecture node, file, or entity to see all the metrics available for that item. See page 208.
- **Metric Definitions:** *Understand* provides a large number of metrics. For a complete and up-to-date list of the metrics, click the  **Show Metrics Definitions** information icon in the Metrics Browser toolbar. You can filter and search this list. Click on a metric for a description of how it is calculated.
- **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 130.
- **Export to HTML:** You can use the Metrics Browser to export the full list of metrics for all architecture nodes and files. See page 210.
- **Export to CSV:** You can choose **Metrics > Export Metrics** from the menus to create a file of all the project metrics in comma-delimited format. See page 210.
- **PERL/C API:** A more advanced way to get existing metrics and calculate new metrics is with the Python and PERL APIs. These provide full access to the *Understand* project. Choose **Help > Python API Documentation** or **Help > Perl API Documentation** for more information. See page 284.

## Project Overview

Choose **Project > Overview** from the menus to see graphs of metrics for the overall project, including the most complex and largest functions, classes (for languages that support classes), and files. Graphs of the project by line type and directory structure are also provided.



gitahead-Windows  
Generated: 5/21/2021 2:08 AM



The names of files, functions, and classes are shown if you hover your cursor over a colored area.

If you choose **Project > Analyze All Files** when the Project Overview is open, statistics about the accuracy and errors found when parsing are added to the top of the Project Overview.



zlib

Generated: 7/18/2021 4:12 PM

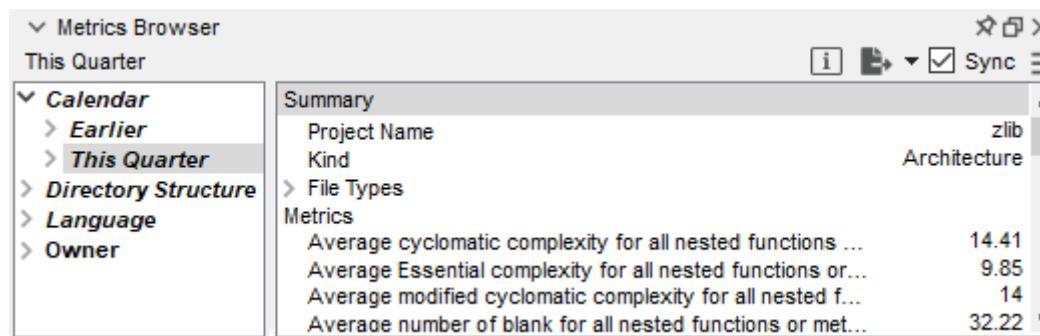
25	67%	113	3	39,852	103	31	495	184
Missing Includes	Parse Accuracy	Errors	Warnings	Lines	Files	Classes	Functions	Subprograms

## Metrics Browser

The Metrics Browser provides an extensive list of metrics for the entire project, any entity, and any architecture node.

To open the Metrics Browser, choose **Metrics > Browse Metrics** from the menus. By default, metrics for the overall project are shown. The top-level node of the “Directory Structure” architecture is selected, which causes the metrics to reflect the entire project.

You can browse the architectures and entities 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.

You can open the Metrics Browser to show metrics for any architecture node, file, or entity in the project by right-clicking the name of an item almost anywhere in *Understand* and selecting **Browse Metrics**.

Click the **Show Metrics Definitions** icon in the Metrics Browser toolbar to learn about the metrics available. See page 209 for details.

Click the **Export Metrics** icon in the Metrics Browser toolbar to save metrics to a file. See page 210 for details. To copy metrics to the clipboard, press **Ctrl+C** or right-click and choose **Copy Selected**. You can hold down the **Ctrl** or **Shift** key to select multiple metrics. Click **Copy All** to copy the full list for the selected node, file, or entity.

If the **Sync** box in the Metrics Browser toolbar is checked, the metrics shown change anytime you select an entity, file, or architecture node in other views.

You can adjust the display by selecting options from the hamburger menu:

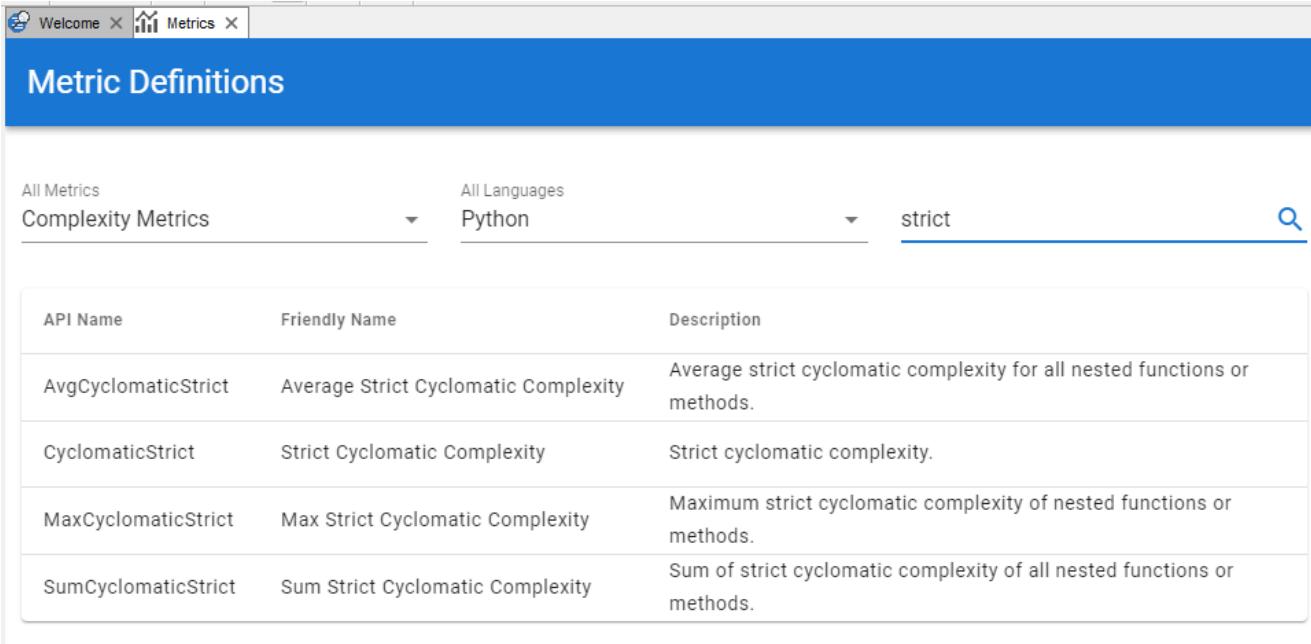
- **Show Detailed Architecture Metrics:** Extends the list of metrics to include complexity and object-oriented metrics in addition to count-related metrics. When disabled, only count-related metrics are shown. For large projects, showing detailed metrics may affect performance when selecting new nodes in the Metrics Browser.
- **Show Summary:** Shows the project name, type of node or entity selected, and a count of the files of various types within the selected node.
- **Show API Name:** Displays the list of metrics using names such as CountLine.
- **Show Friendly Name:** Displays the metrics using short descriptions. The list is sorted differently depending on whether you are viewing it by name or description.

## Metric Definitions

To understand how a metric is calculated, click the  **Show Metrics Definitions** information icon in the Metrics Browser toolbar. This opens the Metrics Definitions window. This window shows both the API name and the “friendly” name of each metric and provides a short description.

Click on any row to open a longer description that includes a diagram to show how the metric is calculated and a list of languages to which this metric applies. Click again to close the long description.

You can search this list by selecting a category of metrics (Complexity, Count, or Object-oriented), a language, and/or a search term.



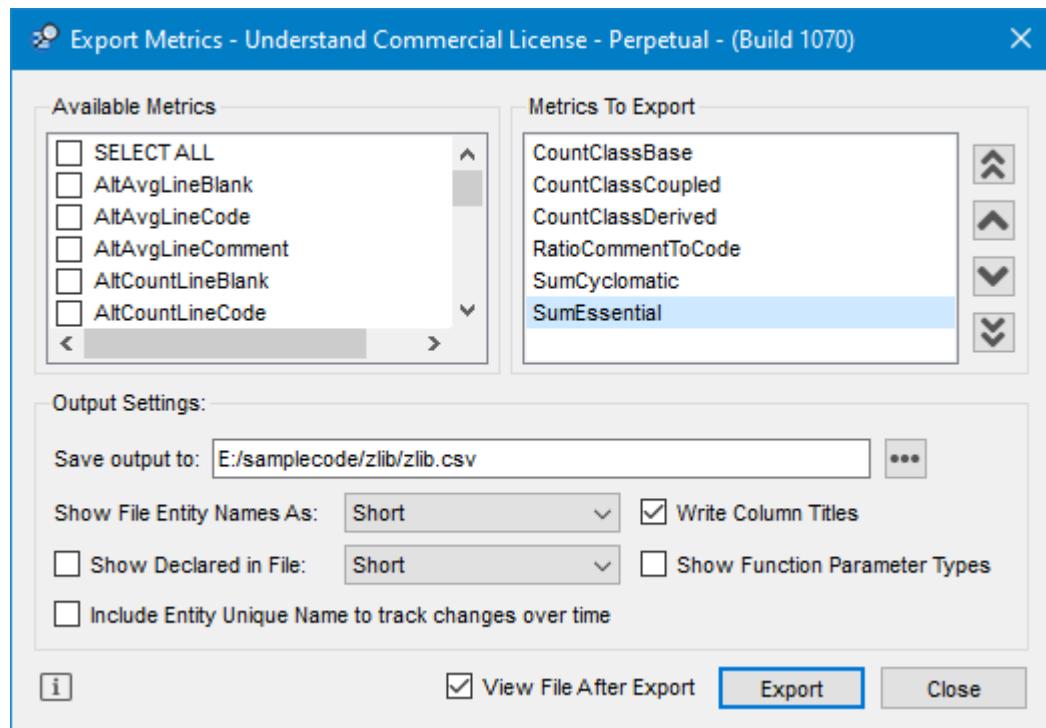
The screenshot shows the 'Metric Definitions' window with a blue header bar. Below the header, there are two dropdown menus: 'All Metrics' (set to 'Complexity Metrics') and 'All Languages' (set to 'Python'). A search bar contains the word 'strict'. The main area displays a table with five rows, each representing a different complexity metric:

API Name	Friendly Name	Description
AvgCyclomaticStrict	Average Strict Cyclomatic Complexity	Average strict cyclomatic complexity for all nested functions or methods.
CyclomaticStrict	Strict Cyclomatic Complexity	Strict cyclomatic complexity.
MaxCyclomaticStrict	Max Strict Cyclomatic Complexity	Maximum strict cyclomatic complexity of nested functions or methods.
SumCyclomaticStrict	Sum Strict Cyclomatic Complexity	Sum of strict cyclomatic complexity of all nested functions or methods.

## Exporting Metrics

The Export Metrics dialog gives you full control over saving metrics to a comma-separated file. To use this dialog, click the  **Export Metrics** icon and choose **Export to CSV**. Or choose **Metrics > Export Metrics** from the menus.

The Export Metrics dialog allows you to select which metrics to export, the order of the metrics in the output, the output file location, and various other options described below. The output includes a row for each function-like, class-like, and structure-like entity. It does not include rows for architecture nodes.



- **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. If your cursor hovers over a metric, you will see a short description of that metric.
- **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.
- **Include Entity Unique Name to track changes over time:** Check this box to add an “Entity\_Uiquename” column with a unique path specification for each entity. This allows you to distinguish between multiple entities that have the same name.

In addition to using the Export Metrics dialog, *Understand* provides the following other ways to save metrics to a file:

- **Copy to a text file:** To copy metrics to the clipboard, press Ctrl+C or right-click and choose **Copy Selected**. You can hold down the Ctrl or Shift key to select multiple metrics. Click **Copy All** to copy the full list for the selected node, file, or entity. Paste the metrics into a file editor.
- **Export selected architecture node to CSV file:** Select an architecture node. Click the down arrow next to the  **Export Metrics** icon and choose **Export Selected Architectures to CSV**. Select a location and filename for the CSV file. You will be asked if you want to open the report, which is a CSV file with a row of metrics for each architecture node selected. All metrics are exported.
- **Export project summary metrics to HTML:** Click the down arrow next to the  **Export Metrics** icon and choose **Export Selected Architectures to HTML**. Select a folder to contain the files. You will be asked if you want to open the report, which is a single page with metrics for the entire project. Only count metrics are exported.
- **Export detailed metrics to HTML:** Click the down arrow next to the  **Export Metrics** icon and choose **Export to HTML**. Select a folder to contain the files. You will be asked if you want to open the report, which is a tree of HTML pages from which you can select any architecture node or file to see metrics for that node or file. Only count metrics are exported.

When prompted for a folder to contain exported metrics in HTML format, the files are actually stored in a folder called “<Project\_name>\_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. After a report is generated, you are asked if you want to view the report. Click **Yes** to open the top-level page, index.html.

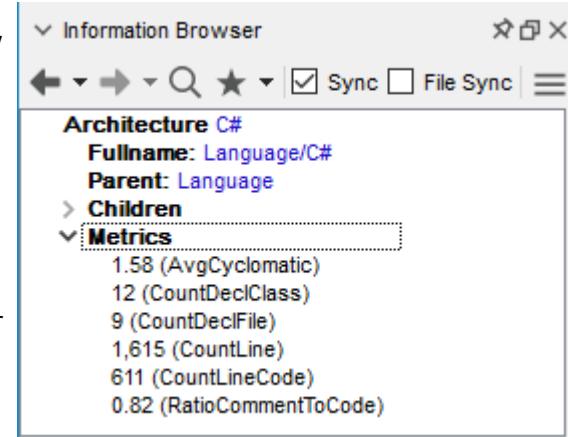
## Metrics in the Information Browser

You can view basic metrics about any entity or architecture node in the Information Browser.

In the Information Browser you can expand the **Metrics** node to see a few simple metrics such as the count of code lines, the ratio of comments to code, and the average cyclomatic complexity for all nested functions or methods.

Click the  **Show Metrics Browser** icon next to the Metrics node to open the Metrics Browser. See page 208 for details.

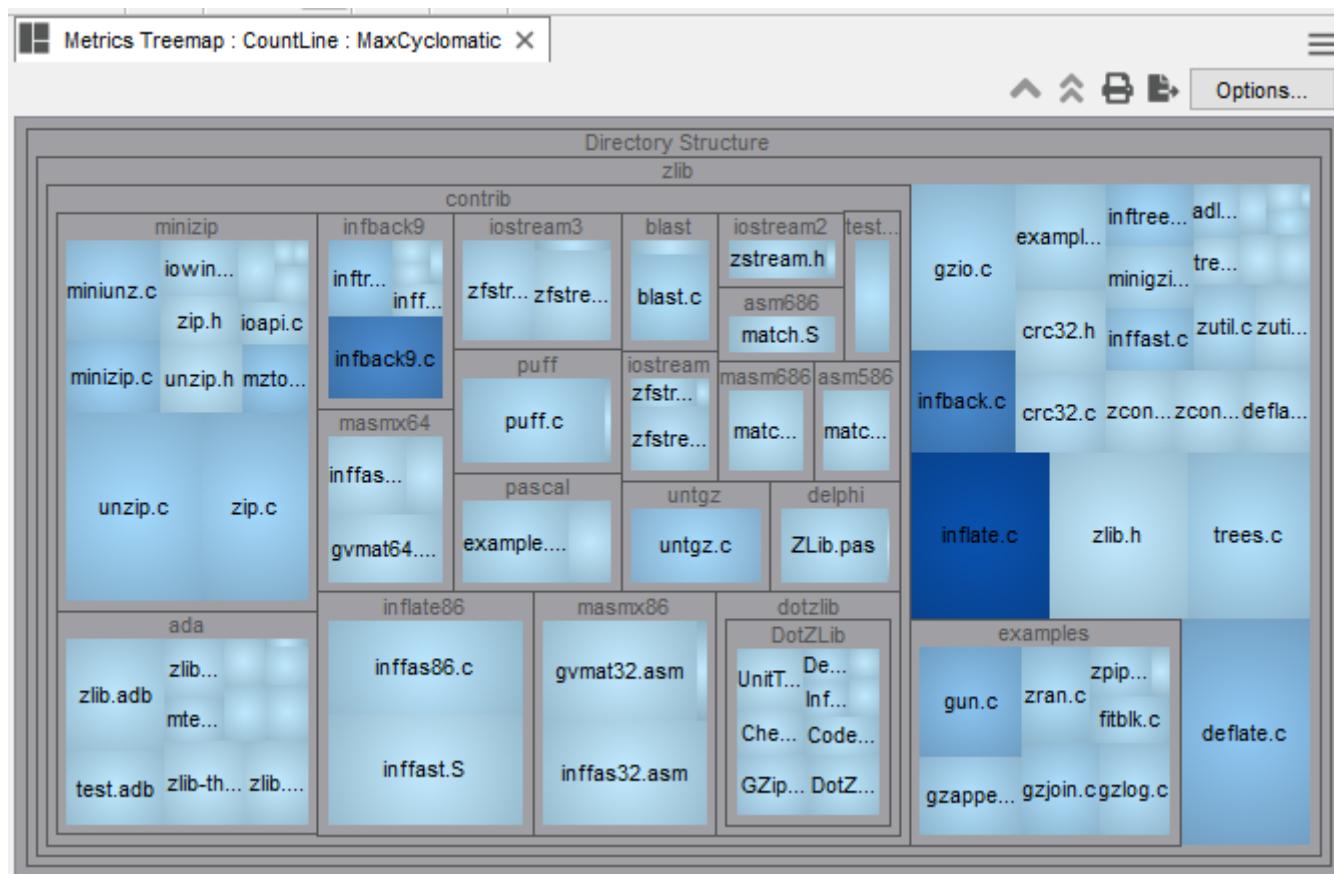
By default, the Metrics node in the Information Browser is collapsed whenever a new entity is viewed. If you want the node to stay expanded, you can click the small arrow to the right of the word “Metrics” and choose **Allow Pre-expansion**. Viewing new entities in the Information Browser may be slower if you enable this option.



## Metrics Treemap

Treemaps show metrics graphically by varying the size of blocks and the color gradient. Each block represents a file, class, or function. 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 quickly learn which files are large and complex vs. files that are large and relatively non-complex.

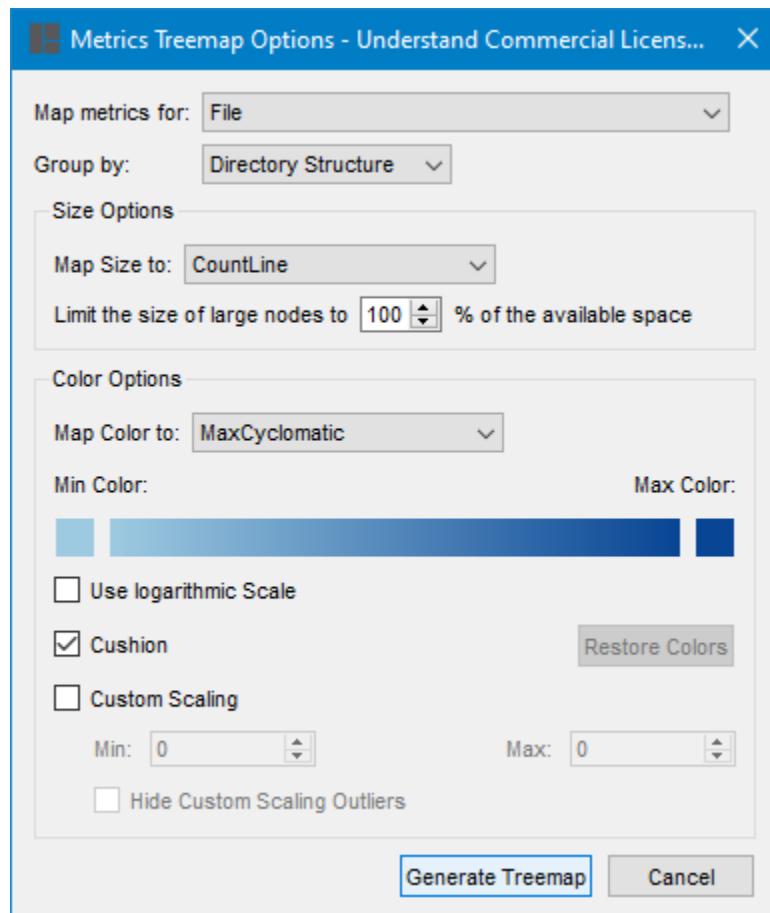


So we learn that `unzip.c` is large, but not particularly complex, while `inflate.c` is large and highly complex.

By default the maps are nested by directory structure. If you have built other architectures, you can use those instead.

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 (including interfaces, packages, and structs), or Functions (including methods, procedures, routines, and tasks).
- 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 (a built-in architecture). Alternately, you can choose to group according to any other defined architecture or no architecture (flat).
- 4 In the **Size Options** area, set **Map Size** to a metric to determine the size of each block. You can also limit the size of the largest block to some percentage of the treemap. You might want to reduce this value 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, set **Map Color** to a metric to determine the color of each block. 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 Check the **Custom Scaling** option and specify **Min** and **Max** values if you want to scale the treemap colors. Nodes for which the metric mapped to the color gradient has a value less than or equal to the Min will have the Min color. Likewise, nodes have the Max color if the metric mapped to the color is greater than or equal to the Max value. This allows easier comparisons between different projects or to allow for before and after pictures that use the same scale. If you want to hide the blocks for nodes outside this range, check the **Hide Custom Scaling Outliers** box.
- 9 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 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.

You can click the  **Print** icon to print the currently displayed treemap diagram.

Click the  **Export to Image File** icon to save the treemap to a PNG, JPG, or SVG file.

Click the **Options** button to reopen the Metrics Treemap Options dialog.

---

## Chapter 10    Using Graphical Views

This chapter covers the graphical views in *Understand* and their options.

This chapter contains the following sections:

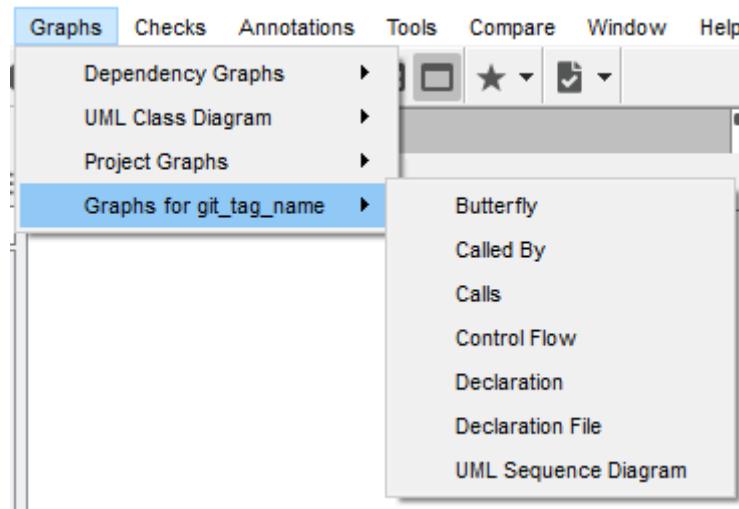
Section	Page
Opening Graphs	217
Graph Variants	220
Controlling Graphical Views	222
Context Menus for Graphs	229
Saving Graphical Views	236
Printing Graphical Views	238
Importing Graphical View Plugins	239

## Opening Graphs

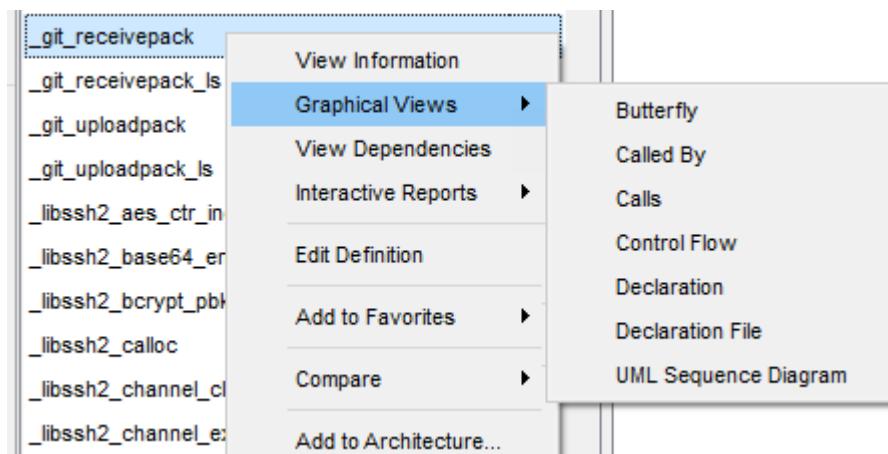
*Understand* provides many types of graphs for visually exploring and explaining your project. The graph categories available change based on the type of entity or architecture node you select. You can select entities or nodes almost anywhere in *Understand*. The list of graph categories is provided for the selected entity or node in the following places:

- **Graphs > Graphs for entityName** list in the top menu bar
- **Graphical Views** submenu when you right-click on an entity
-  **Show Graphic Views Menu** icon in the main toolbar

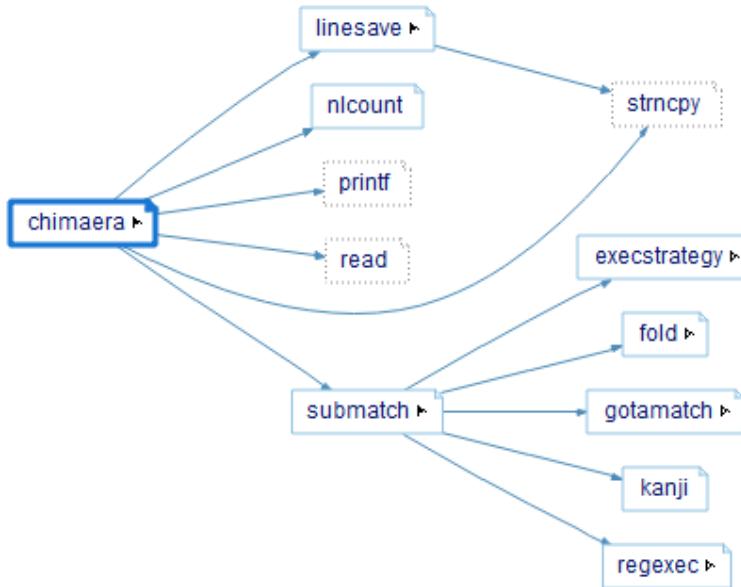
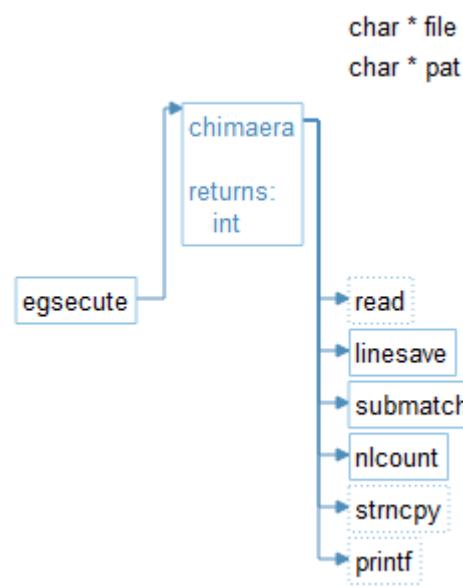
Here are typical graph categories available for a function. For example, the Butterfly graph category provides graphs that show both functions called by and functions that call the selected function.



The same graph categories are listed when you right-click on an entity and choose **Graphical Views**:



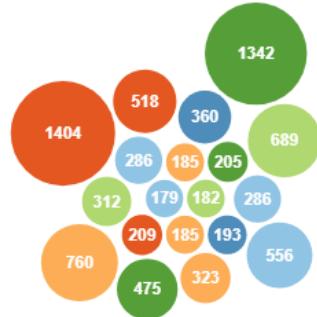
There are many graph types available in *Understand*. Graphs that show the hierarchy of calls and the structure of types are available for all languages that support calls and types. For example, here are two graphs for a C function called “chimaera”:

**Calls Graph (simplified)****Declaration Graph**

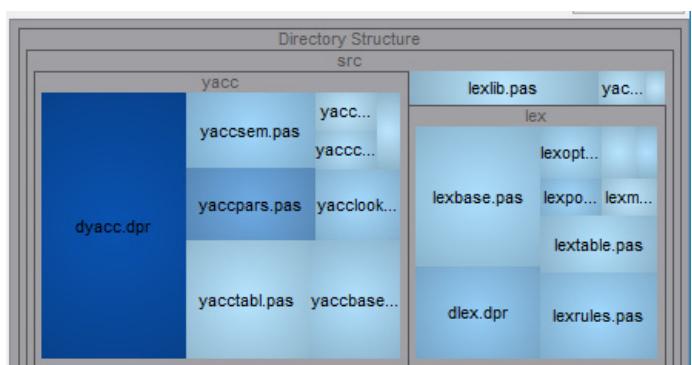
Some graphs apply to certain programming languages. For example, there are Derived Classes and Base Classes graphs for languages that support class inheritance, such as C++. Likewise, there are graphs to show the hierarchy of “include”, “with”, “use”, or similar statements, depending on the keywords used by the source code language.

Other ways to open graphs in *Understand* are as follows:

**Project Overview** graphs can be opened by choosing **Project > Overview** from the menus. These graphs show project-level metrics, such as relative function size and complexity See page 207.



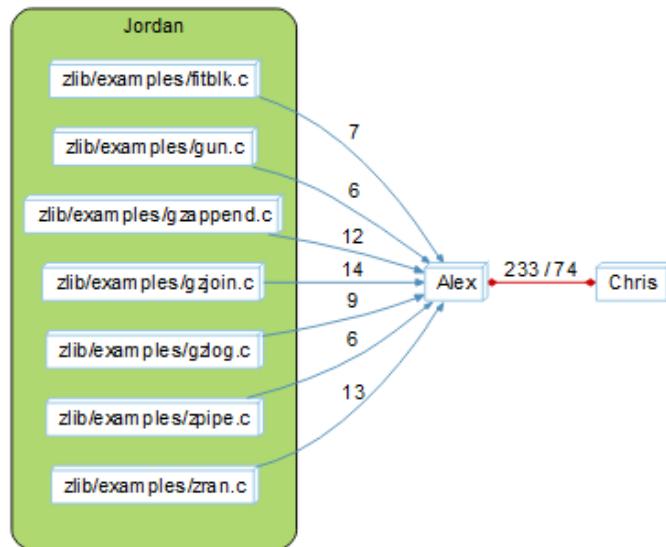
**Metrics treemaps** can be opened by choosing **Metrics > Metrics** **Treemap** from the menus. These are special graphs that can be generated using any two metrics. See page 213.



**UML Class Diagrams** can be opened for an architecture node by choosing **Graphs > UML Class Diagram > nodeName** from the menu bar. These graphs follow the Unified Modeling Language (UML) structure diagram format.



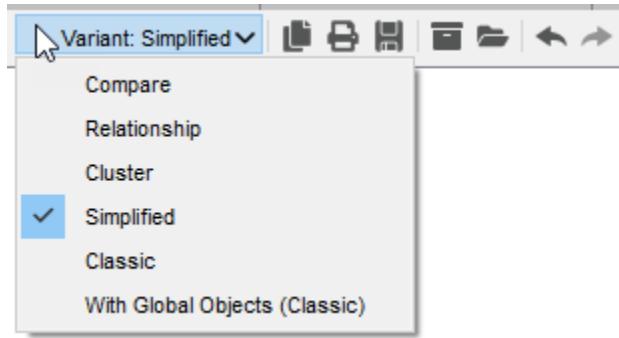
**Dependency graphs** can be opened for an architecture node by choosing **Graphs > Dependency Graphs > architectureName** from the menus. These graphs show relationships between nodes and entities in architectures. See page 201.



**Plugin graphs** can be created using the Perl API. To open plugin graphs you have installed, choose from the **Graphs > Project Graphs** list. See page 284.

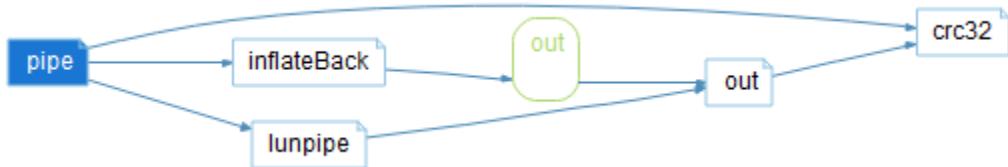
## Graph Variants

Similar graphs are now grouped together for a cleaner and more user-friendly interface. For example, when you open a Calls graph for a function, you may see a list of variant graphs like the following. You can easily switch between the graph variants:

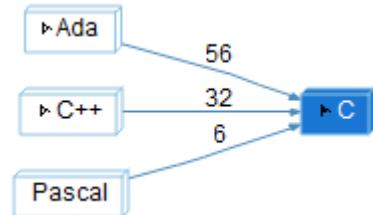


Choose a variant to view that graph. The variants available depend on the entity and graph category you selected, but these are some common variants:

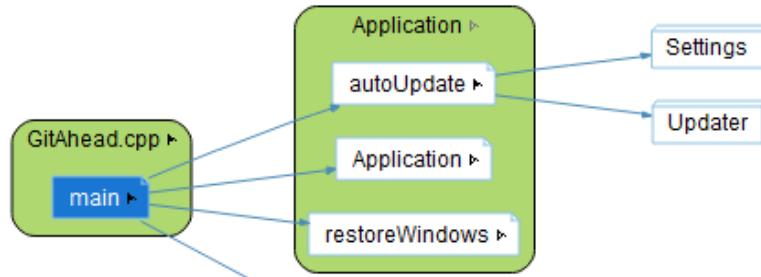
- **Simplified:** This is the default variant. (See page 114 to change this.) For functions, it presents a simple view of the call, call by, or butterfly call tree. For objects or types, it presents a simple view of its members. You can right-click on the graph to control a number of display options. See page 229 for details about these options.
- **Classic:** This is the legacy graphical view. It is similar to the Simplified view, but the right-click options are different. See page 229 for details about these options.
- **With Global Objects (Classic):** This is similar to the legacy graphical view, but includes global objects.
- **Relationship:** This variant shows relationships between any two entities. You will be asked to click on another entity whose relationship to the selected entity you want to find. You can click on the second entity anywhere in the *Understand* interface. The name of the entity you select appears in the “Select a second entity” dialog. This example shows the call relationship from the pipe() function to crc32().



- **Dependencies:** This variant contains nodes drawn as 3D boxes (like those in this figure) that can be expanded to show the nodes they contain by double-clicking on them. You can keep expanding nodes until you get to the entity level.

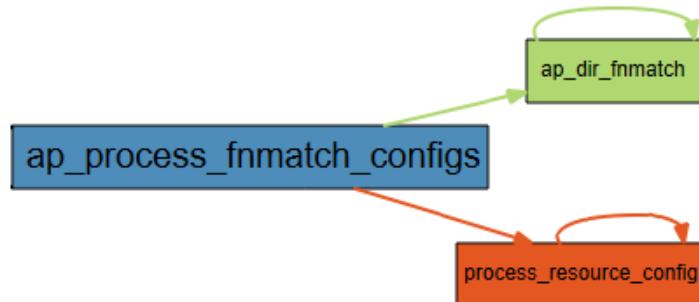


- **Cluster:** This variant provides a view of relationships that is clustered by file. These graphs can be expanded or collapsed by double-clicking on a box in the graph. Cluster graphs can be accessed from the function, class, file, or architecture level.

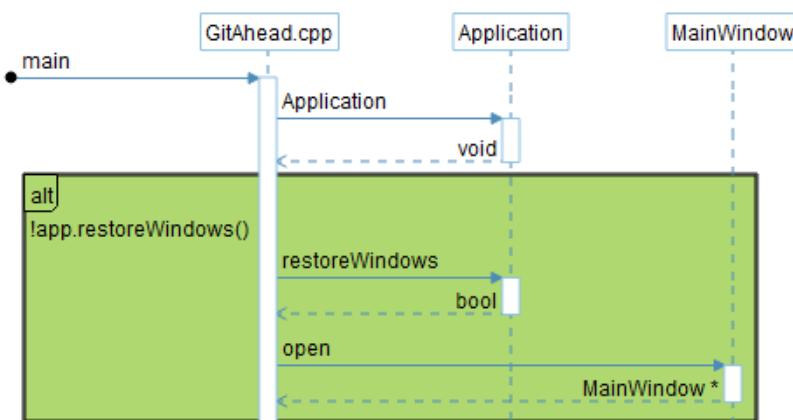


You can customize the display colors, shape fills, shapes, and arrows used in cluster graphs in the **Graphs** category of the **Tools > Options** dialog (page 114).

- **Compare:** This variant shows how changes to a project affect the call structure, data structure, or control flow. To view these graphs, you need to point to a previous version of the same project. See page 266.



- **UML Sequence Diagram:** This variant shows interactions between entities arranged by time sequence. This graph is available for functions and methods that call member methods. Search the [support website](#) for “UML” to find sample UML Sequence diagrams and information about how events are displayed.



- **Custom:** If you have installed plugin graphs, these are available by choosing the Custom variant. See page 285.

## Controlling Graphical Views

The following sections describe controls for using graphical views.

- *Graph Toolbar* on page 222
- *Graph Options* on page 224
- *Mouse Controls* on page 224
- *Classic Context Menu Controls* on page 225
- *Customization Panel Controls* on page 227

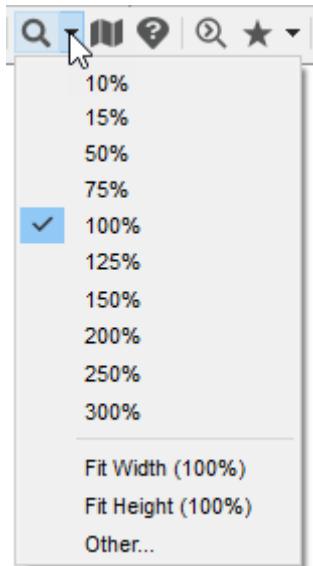
### Graph Toolbar

The toolbar for graphs may contain the following icons. Some graph types do not provide all of these icons. Use the **Graph** category of the **Tools > Options** dialog to control which toolbar icons are shown (page 114).



- **Variant: Simplified** **Variant list:** Select a variant of this graph category to view. See page 220.
- **Copy Image to Clipboard:** Allows you to paste the graph as an image into another application. See page 236.
- **Print Graph:** Opens the Print Options dialog to print the graph. See page 238.
- **Save Graph to File:** Allows you to save a graph as a JPEG, PNG, SVG, Visio, or DOT file. See page 236.
- **Save Graph Customization:** Prompts for a name for the current settings. Settings apply only to the specific graph type and the 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 context menu. Otherwise, type a name for your current settings and click **Save**.
- **Load Graph Customization:** 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:** Undo your last change to graph customization.
- **Redo:** Redo the last change you undid to graph customization.
- **Restore Defaults:** Restores the graph to the original settings.

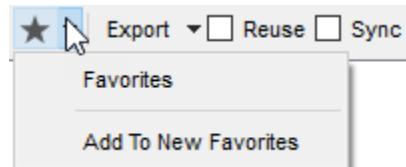
- **Zoom:** You can zoom in or out using this drop-down list or the mouse scroll wheel. If you do not want the mouse scroll wheel to control the zoom level, you can disable that behavior in the Graphs category of the Options dialog (page 114).



- **Show Mini-Map:** Toggle this icon on to see a small map that shows the full graph. For large graphs, this map helps to know where to navigate within the graph using the scroll bars. Drag or click within the mini-map to move to a new location in the graph.
- **Show Legend:** Legends are available for some graph types. If this icon is shown in the toolbar, click the icon on to show a graph legend. The legend identifies the shapes and arrow styles used in the graph. The legend can be dragged to reposition it within the graph.
- **Search Graph:** Click icon or press Ctrl+F to display the incremental search bar at the bottom of the graph. You can use this bar that 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 163.



- **Favorites:** Select the name of a favorites list to add this graph to that list. Select **Add To New Favorites** to create and name a new favorites list with a link to this graph. See page 142.



-  **Reuse**: This checkbox controls whether a graphical view is reused or a new tab is opened when another graphical view is requested. This box is unchecked by default. At most one graphical view can have the **Reuse** box checked at any time; the box is automatically unchecked in other tabs when you check this box.
-  **Sync**: This checkbox controls whether this graphical view changes when a different entity is selected in the Project Browser, Entity Filter, and other areas 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.
-  **Release/Capture Window**: Toggle this icon to open the graph in its own separate window. This is useful when viewing larger graphs.
-  **Configure Graph Settings**: Open the Graphs category of the **Tools > Options** dialog (page 114).
-  **Progress**: This icon is shown to the right of the Configure icon if a graph is currently being updated.
-  **Graph Settings**: The hamburger menu provides the same commands as right-clicking on the background of the graph (page 229).

---

## Graph Options

See page 114 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 over the mouse over a graph or double-click on a node.

To control the font used in graphs, set the **Application Font** in the **General** category of the **Tools > Options** dialog. See page 94.

---

## Mouse Controls

Right-clicking on the background of a graph or clicking the hamburger menu for a graph shows controls that apply to the entire graph (page 229).

Right-clicking on an entity in a graph shows full context menus and controls for that entity. The **Entity** submenu contains the commands you see when you right-click on the entity elsewhere in *Understand*. The **Graph** submenu provides commands to control the appearance of the graph (see page 229). Other commands in the context menu are specific to the type of graph.

In some graph variants, right-clicking on an edge (line or arrow) provides a list of the references that constitute the edge. Choose an item from the list to view the source code for this relationship. You can limit the length of this list as described for the **Graphs** category in the **Tools > Options** dialog on page 114.

Single-clicking on an entity in a graph shows information about the entity in the Information Browser if the **Sync** box is checked in the Information Browser.

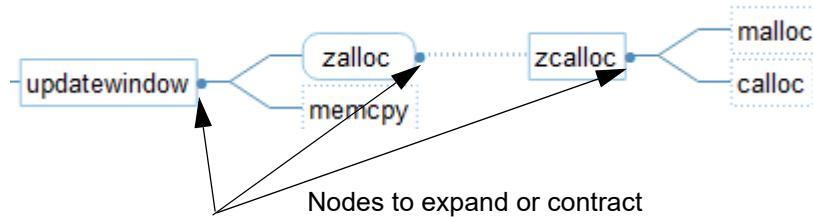
You can select one or more nodes in a graph by holding down the Ctrl key and using your mouse to drag a rectangle over the nodes you want to select.

Double-clicking on an entity in a graph expands that entity if possible for most types of graphs.

## Classic Context Menu Controls

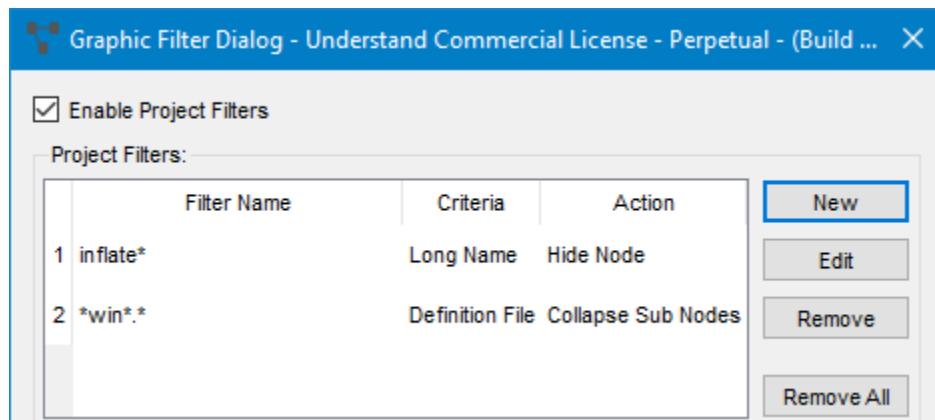
In Classic graph variants and some other graph types, commands besides the **Entity** submenu and **Graph** submenu may be provided for actions like the following:

**Expanding hierarchy:** In Classic graph variants, you can expand and contract tree views by clicking the blue circle to the right of a node. Right-click on a node and choose **Open Node**, **Close Node**, or **Close Other Nodes**. 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. Hold down the Ctrl key to select multiple nodes if you want to keep multiple nodes open when using **Close All Nodes**.

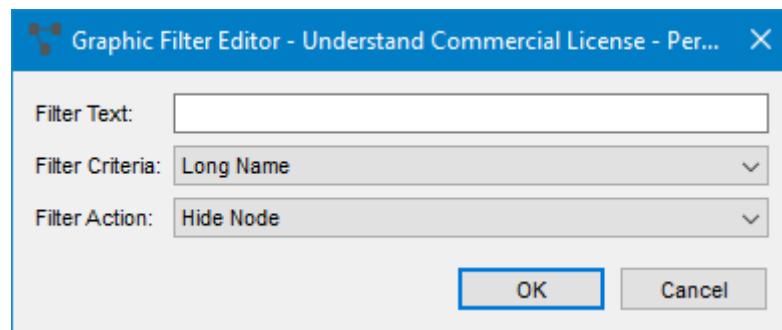


**Filtering graph contents:** In Classic graph variants, you can apply filters to hide certain entities in some graphical views. To create such a filter, follow these steps:

- 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, check the **Enable Project Filters** box.
- 3 Click **New**. This opens the Graphic Filter Editor dialog.



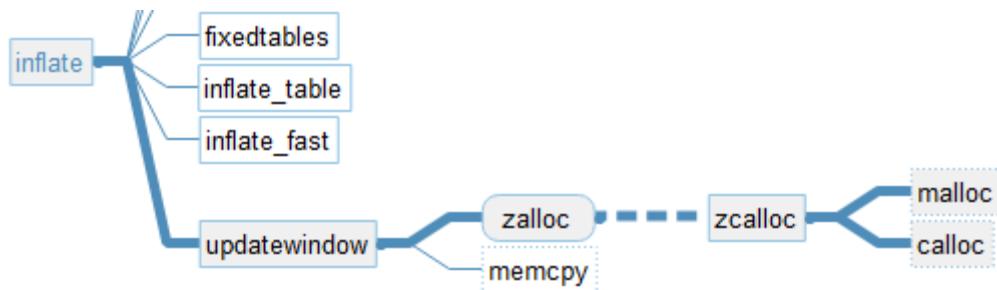
- 4 Type a filter in the **Filter Text** field. For example, use gr\* to match entity names beginning with gr. Filters are case-sensitive.
- 5 In the **Filter Criteria** field, select whether to compare the filter to long names, definition files, the type text of entities, or parameter text. For example, 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 graph.
  - **Hide Sub Nodes:** The item that matches the filter is shown, but any subnodes of these items are removed from the graph.
  - **Collapse Sub Nodes:** Any subnodes of items that match the filter are collapsed in the graph. An icon is shown after the node to indicate that there are subnodes. Items that match the filter are shown.
  - **Show Only:** Only items that match the filter are included in the graph.
- 7 Click **OK** to add the filter to the project.

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 Graphic Filter dialog or by right-clicking on any graphical view and choosing **Disable Graphic Filters** from the context menu.

You can also create filters by right-clicking on an entity in a graphic and choosing one of the filtering options. The options allow you to quickly filter out entities with that name, with that type, or in that file.

**Path highlighting:** In Classic graph variants, to highlight the path to one or more entities in a tree view (such as a Callby view), select the entity and right-click. In the context menu, choose **Highlight Path**. Hold down the Ctrl key to select multiple entities for path highlighting.

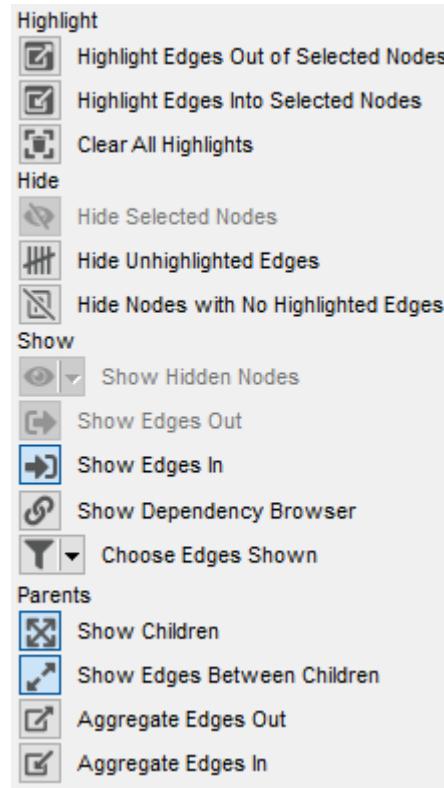


## Customization Panel Controls

The panel to the right of graphs such as dependency and cluster graphs lets you hide, show, and highlight parts of the graph. The commands in the graph customization panel are different for different graph types. Commands are active depending on what is selected in the graph and whether the selected nodes have children and/or edges coming in or out.

Some of the commands are as follows:

- **Highlight Edges Out of Selected Nodes.** Causes relationships (“edges”) from the selected node(s) to other nodes to be highlighted. Internal Dependency graphs let you highlight such edges; other types of dependency graphs let you show or hide such edges.
- **Highlight Edges Into Selected Nodes.** Causes relationships from other nodes to the selected node(s) to be highlighted. Internal Dependency graphs only let you highlight such edges; other types of dependency graphs let you show or hide such edges.
- **Clear All Highlights.** Click this button to turn off all node and edge highlighting.
- **Hide Selected Node(s).** Removes all the nodes that are currently selected from the graph and reorganizes the graph as needed. (You can later restore the hidden nodes by clicking Show Hidden Nodes.)
- **Hide Unhighlighted Edges.** If you check this box, all edges that are not highlighted are hidden, and the graph is reorganized as needed to omit those non-highlighted relationships. You can use this only if you have turned on some highlighting of edges.
- **Hide Nodes With No Highlighted Edges.** If you check this box, all nodes that do not have a highlighted edge coming into or out of it are hidden, and the graph is reorganized as needed to omit those nodes. You can use this only if you have turned on some highlighting of edges.
- **Show Hidden Nodes.** If you click this button, any nodes that have been hidden using one of the “Hide” buttons are restored. This button does not expand any nodes that have been contracted to hide child nodes. If you have hidden nodes, you can select entities from the **Hidden Nodes** drop-down list to redisplay those nodes.
- **Show Edges Out.** Causes arrows to be drawn between the selected node and any other nodes as appropriate. If you remove the display of arrows, the graph is reorganized to hide these relationships.

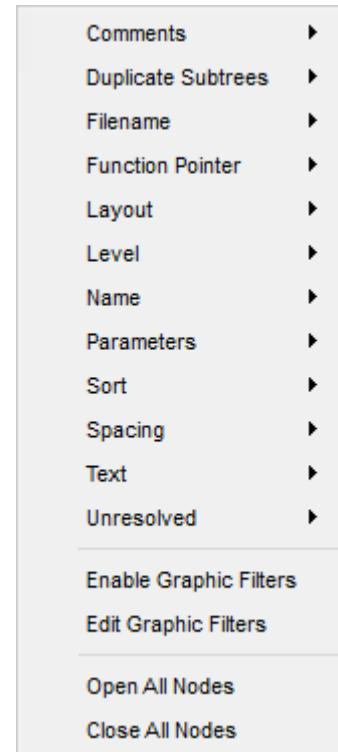


- **Show Edges In.** Causes arrows to be drawn between the selected node and any other nodes as appropriate. If you remove the display of arrows, the graph is reorganized to hide these relationships.
- **Show Dependency Browser.** This button opens the Dependency Browser (page 135) for the most recently selected node or relationship in the graph. If you check the **Sync** box in the Dependency Browser, it shows details about any relationship you select in the graph, and the two nodes connected by the relationship are highlighted in the Dependency Browser.
- **Choose Edges Shown.** You can choose the types of dependencies you want shown in the graph from this list. The dependency types available include Inits (initializes), Sets, Uses, Calls, and Modifies. By default, all types of dependencies are shown.
- **Show Children.** Causes any child nodes of the selected node to be displayed. This is the same as double-clicking on a node to expand it.
- **Show Edges Between Children.** Causes arrows to be drawn between the selected 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 Edges Out.** Causes arrows coming from the selected node's children to be drawn coming from the node, and arrows with the same target from multiple children to not be repeated. Toggle this off to cause separate arrows to be drawn from the individual child nodes.
- **Aggregate Edges In.** Causes the arrows going to the selected node's children to be drawn as going to the node, and arrows to multiple children are not repeated. Toggle this off to cause separate arrows to be drawn to the individual child nodes.

## Context Menus for Graphs

When you right-click on a graph, a number of submenus allow you to choose how to configure and display the graph. The submenus available depend on the type of graph. The sections that follow describe many, but not all, of those submenus. They are listed here in alphabetic order.

- The **Aggregate Nodes by** menu controls how nodes are grouped in cluster graphs. Options are by function, by class, by file, and by architecture. The default is to group entities by file.
- The **Aggregate by Architecture** menu controls how nodes are grouped. Options are the defined architectures. The default is off.
- The **Allow Call Expansion** menu controls whether called functions can be expanded. If this option is on, expandable calls are shown as a 3D box. The default is off.
- The **Annotations** menu controls whether annotations attached to the entity are not shown (Off, the default), only the most recent annotation is shown (First), or all annotations are shown.
- The **Architecture Name As** menu controls how architecture names are displayed. The options are short, relative to the project, and long names. The default is relative.
- The **Background Colors** menu controls whether the boxes for entities are filled in using the colors in the legend. The default is off.
- The **Base Classes** menu controls whether declaration graphs include base classes if applicable.
- The **Calls Depth** menu controls how many levels of calls are displayed. The default is 3 levels.
- The **Called By Depth** menu controls how many levels of callbys are displayed. The default is 3 levels.
- The **Called by** menu controls whether program units that call the current entity are shown in declaration views.
- The **Cluster** menu controls whether statements in a group such as the “if” or “else” branch of a conditional statement are clustered in UML diagrams. The default is on.
- The **Collapse** menu controls whether to combine statements into a single box if there are no decision points between them in a control flow graph. The default is on.
- The **Comments** menu controls whether to show any comments associated with an entity. The default is off.



- The **Constants** menu controls whether to show constants in Declaration views. The default is on.
- The **Debug** menu controls whether to show details about each item in control flow graphs. In order, the detail information is: nodeID, nodeKind, startLine, startCol, endLine, endCol, endNode, commaSeparatedListOfChildren. The default is off.
- The **Default Members** menu controls whether declaration views show default members of the class.
- The **Define/Declare** menu controls whether define and declare relationships are shown as edges. It is available in Object References and Compare Object References graphs. The default is off.
- The **Depended On By Depth** menu controls the number of levels of entities that depend on this entity shown.
- The **Depends On Depth** menu controls the number of levels of entities this entity depends on shown.
- 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.
- The **Dependents** menu controls whether files that are dependent on the current C file are shown in File Declaration graphs. The default is on.
- The **Derived Classes** menu controls whether declaration graphs include derived classes if applicable.
- The **Duplicate Subtrees** menu controls whether multiple occurrences of the same sub-tree 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.
- The **Edge Labels** menu controls whether labels are shown for lines and arrows between entities.
- The **Entity Name Format As** menu controls whether long or short entity names are displayed. The default is short.
- The **Expand Macros** menu controls whether macros are expanded in control flow graphs if you have enabled the **Save macro expansion text** option in the C++ project configuration (page 62).
- The **Extended By** menu controls whether declaration views show classes by which the selected class is extended.
- The **Extends** menu controls whether declaration views show classes that the selected class extends.
- The **External Functions** menu controls whether 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. The default is on.
- The **Filename** menu controls how filenames are displayed in views. It is available for both declaration and hierarchy views.
  - 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.
- Relative: Where filenames are relevant, the file path relative to the project is shown in square brackets.
- The **Filter** menu controls whether to hide implicit actions, such as “endif” in control flow graphs. The default is on.
- The **Function Pointer** menu controls whether function pointers are displayed as invocations in the Call and CallBy trees.
- The **Globals** menu controls whether to show globals in Declaration views. The default is on.
- The **Global Objects** menu controls whether to show global objects.
- The **Implements** menu controls whether declaration views show entities that the selected entity implements.
- The **Implemented By** menu controls whether declaration views show entities by which the selected entity is implemented.
- The **Imports** menu controls whether declaration views show entities imported by the current entity.
- The **Included By** menu controls whether files that include the header file are shown in a Header File Declaration view. The default is on.
- The **Includes** menu controls if include files are drawn on file declaration diagrams (C file, Header file). The default is on.
- The **Include Depth** menu controls how many levels of file includes are displayed. The default is 3 levels.
- The **Includeby Depth** menu controls how many levels of file includebys are displayed. The default is 3 levels.
- The **Include Global Objects** menu controls whether global objects are included in cluster call graphs. The default is off.
- The **Include Standard Libraries** menu controls whether standard libraries are included in the UML diagram.
- The **Include Unresolved** menu controls whether unresolved objects are included in the graph. The default is off.
- The **Include Virtual Edges** menu controls whether to show override and overriddenby edges. If you want to change the color of virtual edges, choose **Tools > Options**. In the **Graphs** category, go to the Cluster Graph Styles section, and in the Edges list, modify the display format for Virtual Calls.
- The **Inherits** menu controls whether declaration views show entities that the selected entity inherits.
- The **Inherited By** menu controls whether declaration views show entities inherited by the selected entity.

- The **Intrinsic Functions** menu controls whether intrinsic functions (for example, cos and sin) are displayed or hidden.
- The **Invocations** menu controls whether procedures and functions called by the current procedure or function are shown in Declaration views.
- The **Layout** menu controls how to arrange the graph. It is available only in hierarchy and control flow graphs. For control flow graphs, the options are Horizontal and Vertical. For hierarchy graphs, 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.
- The **Legend** menu controls whether a legend for the graph is shown. The legend identifies the shapes and arrow styles used in the graph. The default is off.
- The **Level** menu controls the number of levels to be traversed when laying out a hierarchical or dependency graph. The default value is “All Levels”. Values of 1 to 5 may be set. This is available only in hierarchy views.
- The **Locals** menu controls whether local items are shown in Declaration views. The default is on.
- 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.
- The **Modified Entities** menu controls what types of modifications are shown in comparison graphs. The default is to ignore changes to comments, spaces, and newlines.
- 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\_Io.Put is the fully specified name.
- Put is the Short Name

Longer versus shorter names can alter the layout of pictures substantially.

- The **Objects** menu controls whether to show objects in Declaration views. The default is on.
- The **Operators** menu controls whether entities that are operators are shown in the Callby, Declaration, Declaration Tree, and Invocation views.
- The **Overrides** menu controls whether function overrides are shown.
- The **Parameters** menu controls whether parameters are shown in hierarchical views. This menu is available for any hierarchical graphical view (invocation and callby). The default is off; turning this on can make hierarchical pictures much bigger.

- The **Passive** menu controls whether passive nodes are shown in control flow graphs. The default is on.
- The **Private Members** menu controls whether declaration views show private members of the entity.
- The **Protected Members** menu controls whether declaration views show protected members of the entity.
- The **Public Members** menu controls whether declaration views show public members of the entity.
- The **Random Edge Color** menu controls whether edges are displayed using random colors. The default is to display all edges in the same color.
- The **References** menu controls whether filenames and numbers of references are shown in comparison graphs. The default is to show all reference information.
- The **Renames** menu controls whether declarations that are renames are shown in Declaration views. The default is to show rename declarations.
- The **Routines** menu controls whether to show routines (procedures, functions, etc.) in Declaration views. The default is on.
- The **Show Class Details** menu controls whether details about each class are shown.
- The **Show Related Classes** menu controls whether classes related to this class are shown.
- The **Show Solo Classes** menu controls whether comments associated with statements are shown in control flow graphs. The default is off.
- The **Show Edge Labels** menu controls whether the number of relationships between two entities is shown as a label. For bi-directional call relationships, the number is the total number of relationships in both directions. The default is off.
- The **Show Edges Between Children Default** menu controls whether to show inter-child edges initially for nodes that are expanded. Note that changing this setting only affects nodes that are using the defaults; You may need to click the **Restore Defaults** icon to affect the entire graph. The default is off.
- The **Show Entity Name** menu controls whether to shows the name of the entity in the Start box at the beginning of a control flow graph. You can choose to hide, show, or show the name with parameters. The default is to hide the name.
- The **Show Finally-Block Flows** menu controls whether to show edges representing exceptional exits from a try-catch block in languages like Java and C# in control flow graphs. The default is on.
- The **Show Labels** menu controls whether to show text for edges (for example, yes/no) and start block in control flow graphs. The default is on.
- The **Show Node Children By Default** menu controls whether to open nodes by default when you open a cluster graph. For example, all functions within files will be shown by default if this option is enabled when you open the Cluster Callby graph for a file. Note that changing this setting only affects behavior the next time you open the graph. The default is off.

- The **Show Returns** menu controls whether UML diagrams show function returns. The default is on.
- The **Show Self Calls** menu controls whether UML diagrams show function self-calls. The default is on.
- The **Show Source Code** menu controls whether source code is hidden, shown, or shown only for decisions in control flow graphs. The default is on.
- The **Show Version Differences** in comparison graphs controls whether added, removed, or both types of differences are shown. The default is both.
- The **Show Unresolved Entities** menu controls whether UML diagrams show unresolved entities. Options are to hide, show, or show in a separate column. The default is hide.
- 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.
- The **Spacing** menu lets you choose to change the space between boxes. You can choose compact, small, normal, wide, or extra wide.
- The **Sql** menu lets you specify whether SQL entities should be shown in graphical views. This option is on by default.
- 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). The default is on.
- The **Styled Labels** menu controls whether to highlight keywords, comments, and strings in source code shown in control flow graphs. The formats defined in the **Editor > Styles** category of the Understand Options dialog (page 111) are used. The default is on.
- 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. This is 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.
- The **Type or Typetext** menu controls whether to show type information in the graph.

- The **Unknown** menu controls whether entities should be shown if they are used in the source, but are never declared or defined.
- The **Unresolved** menu controls whether entities should be shown if they have been declared but not defined. For example, an entity may be declared in a header file, but never defined in the source. Unresolved include files are those that are 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,
- The **Use Class Grouping** menu controls whether entities are grouped by the class that contains them. The default is off.
- The **Usedby** menu tells Declaration views whether to show items that use this item.
- 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.
- The **Variables** menu controls whether to show globals in Declaration views. The default is on.
- The **Virtual Calls** menu controls whether to show calls through a virtual function to its derived function. The default is off.
- The **Withs** menu controls on Declaration views of compilation units (packages, tasks, separate procedures, etc...) if Withs are drawn. The default is on.
- The **With Bys** menu controls whether who “Withs” a given compilation unit is shown in declaration views. The default is on.

## Saving Graphical Views

*Understand* offers various ways to export your graphical views. The toolbar for each graphical view provides the following icons for copying and printing graphs.

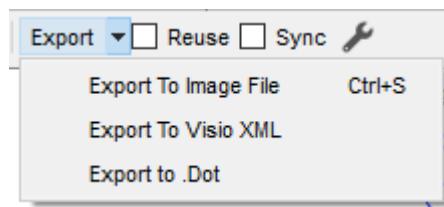


In addition to printing (page 238), you can save graphical views as JPEG, PNG, SVG files (page 236), Visio XML files (page 236), and DOT files (page 237). The first three formats are common graphics formats.

For cluster graphs, you can save and load customized graph settings (page 222).

### Saving Views to Files

To save a graphical view in one of the following formats, use the **Export** drop-down on 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.

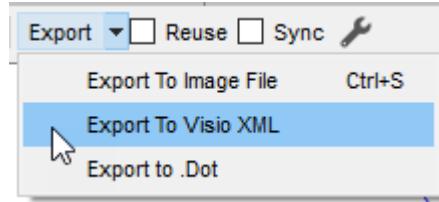
Note that if the graph would result in an image larger than 200 MB, the graph will be resized to a smaller size.

### 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 Visio installed in order to save a graphical view as a Visio XML file.

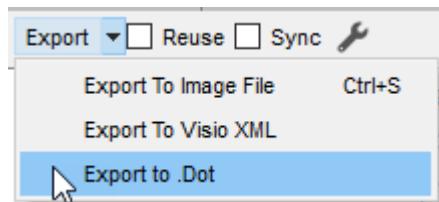
To save a Visio XML file, use the **Export** drop-down the graphical view toolbar to choose the **Export to Image File** option. In the Export dialog, choose a location and filename for the view. The file extension for Visio XML files is \*.vdx.



#### 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

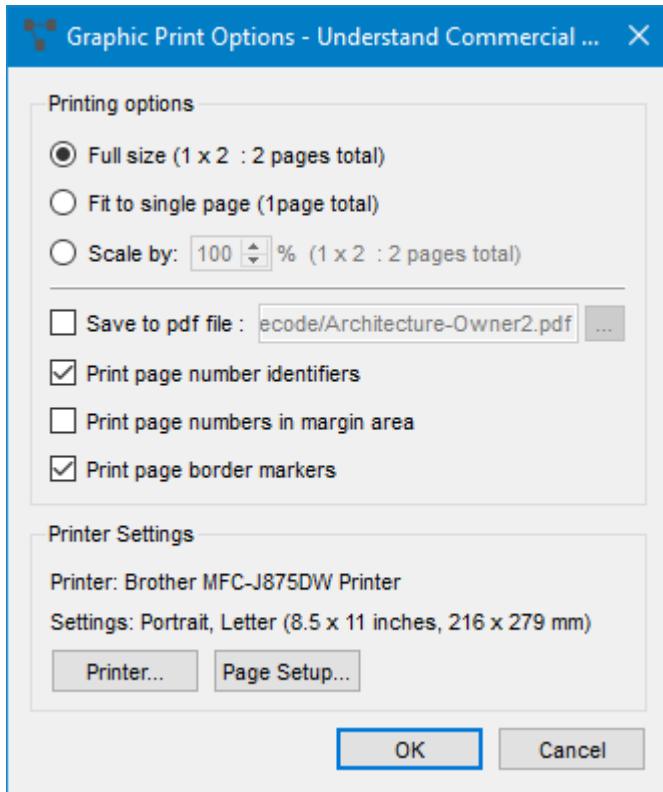
*Understand* has these printing modes:

- Source File printing sends a text file to the printer using 66 lines of source per page. See *Printing Source Views* on page 191.
- Graphical view printing provides options for how to fit the image to a page. See *Graphical View Printing* on page 238.

### 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.

(1,3)

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.

## Importing Graphical View Plugins

Plugins are special Perl scripts that can be imported to provide customized graphical views. Several graphical view plugins are available in the API/Plugins category on the [Support website](#).

See *Plugin Graphs* on page 285 for how to install and run plugin graphs.

Be aware that generating graphs for large projects can often be resource intensive, and in some cases the system can be non-responsive for a long period of time while the graphs are generated.

This chapter explains how to use CodeCheck to find places where your code does not conform to various standards.

This chapter contains the following sections:

<b>Section</b>	<b>Page</b>
About CodeCheck	241
Running a CodeCheck	242
Configuring Checks	243
Selecting Files to Check	245
Viewing Results	246
Ignoring or Excluding Violations	250
Exporting CodeCheck Results	254
Writing CodeCheck Scripts	255

## 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.

Results	Entity	Line	Column	CheckID
290 Number of Results: 290				
147 > C:\SciTools\samples\fastgrep\fastgrep\egrep.c				
1 < C:\SciTools\samples\fastgrep\fastgrep\regerror.c				
↳ 1 Unused Parameter	s	5	6	RECOMMENDED_14
3 void				
4 regerror(s)				
5 char *s;				
6 {				
7 #ifdef ERRAVAIL				
67 > C:\SciTools\samples\fastgrep\fastgrep\regexp.c				
2 < C:\SciTools\samples\fastgrep\fastgrep\regexp.h				
> 1 No comment with variable declaration	startp	9	8	RECOMMENDED_1E
> 1 No comment with variable declaration	endp	10	8	RECOMMENDED_1E

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.

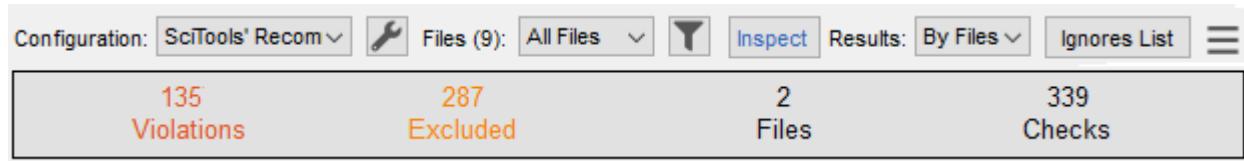
If you want to perform custom checks, you can create your own checks using Perl. For example, you can create a check to find lines longer than 80 characters or filenames that begin with a number.

CodeCheck validation suites are available in the CodeCheck category on the [support website](#).

To configure and run dependency checks on an architecture, see page 203.

## Running a CodeCheck

To open the CodeCheck tool, choose **Checks > Open CodeCheck** from the menus. The CodeCheck window has the following toolbar:



You can quickly run CodeCheck and view the results by following the steps below. For each step, there are ways to further customize the behavior of CodeCheck; these are described in detail in the referenced sections.

- 1 Select a set of checks to run using the **Configuration** drop-down list. For example, you can choose “SciTools’ Recommended Checks” or “AUTOSAR” checks. Or, click the wrench icon and choose checks or configure a set of checks as described in *Configuring Checks* on page 243.
- 2 Select which files to check using the **Files** drop-down list. You can choose All Files, only files modified since the last CodeCheck run, only changes to a Git repository that are uncommitted. To create a custom filter, click the filter icon or select **Customize** from the drop-down list and create a filter as described in *Selecting Files to Check* on page 245.
- 3 Click the **Inspect** button to run the selected checks on the selected files. For large projects, the checks may take significant time to perform.  
Choose **Checks > Re-Run Configuration\_Name** to run the most recently run CodeCheck configuration. If you have made changes to any files, you will be asked if you want to re-analyze the changed files before running CodeCheck.  
Choose **Checks > Analyze Changes and Re-Run Configuration\_Name** to run the **Analyze Changed Files** command automatically and then run the most recently run CodeCheck configuration.
- 4 Use the **Results** drop-down to change how the results are displayed. The hamburger menu also provides ways to customize the results display. You do not need to re-run the checks to change the display. See *Viewing Results* on page 246 for details on ways to view results.
- 5 To hide some of the violations from the results, use the ignore features described in *Ignoring or Excluding Violations* on page 250. The hamburger menu also provides ways to handle ignored violations.
- 6 Use the CodeCheck hamburger menu to control the results display. For example, **Auto Load Last Results** controls whether CodeCheck automatically displays results from the last CodeCheck performed when you reopen it. **Release Window** controls whether the CodeCheck window is contained within the Understand window. See *Viewing Results* on page 246 and *Ignoring or Excluding Violations* on page 250 for details about other hamburger menu commands.
- 7 To export results from CodeCheck, see *Exporting CodeCheck Results* on page 254.

## Configuring Checks

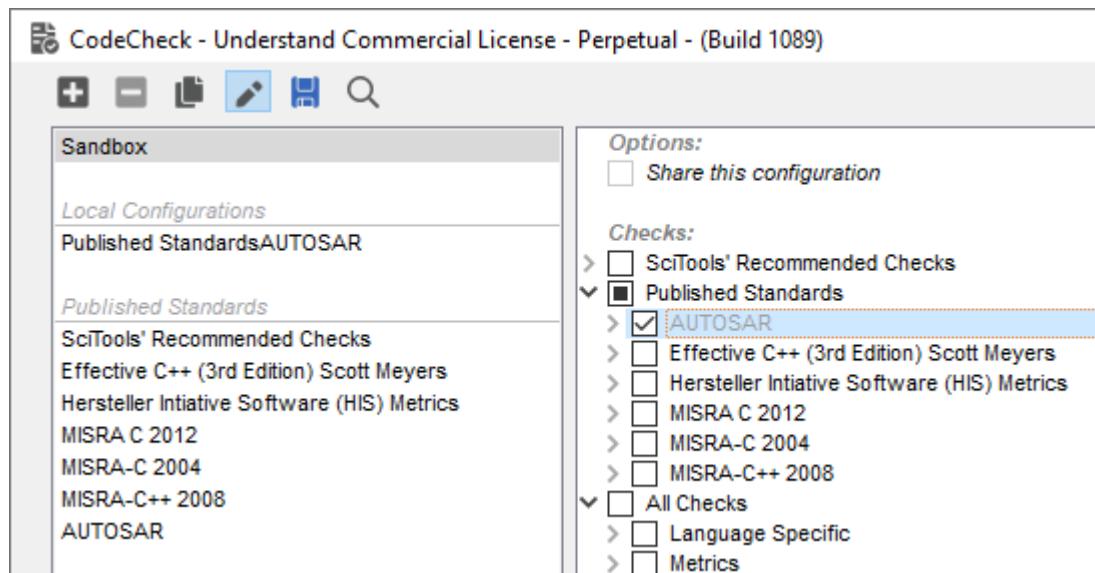
In the **Configuration** drop-down list, you can select any of the built-in sets of checks. For lists of implemented checks, see [the provided spreadsheet](#). The following sets are pre-configured to provide quick checks. These configurations are not editable.

- SciTools' Recommended Checks: Recommended checks for your source code languages. These are standards violations that we feel are most serious.
- Effective C++ (3rd Edition) Scott Meyers
- Hersteller Initiative Software (HIS) Metrics
- AUTOSAR (C++14)
- MISRA-C 2012
- MISRA-C 2004
- MISRA-C++ 2008

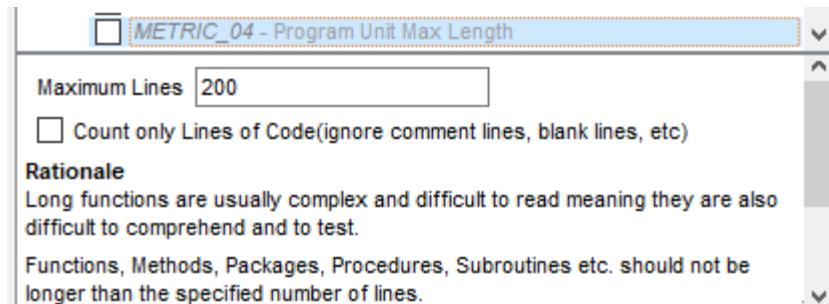
To configure sets of checks that meet your specific needs, click the wrench icon or choose **Checks > Checks Selection** from the menus to open the CodeCheck dialog. This dialog lets you create your own CodeCheck configurations either based on existing configurations or from scratch.

To create or modify a configuration:

- 1 Click the plus icon. Or, select a configuration and click the duplicate icon. You can also right-click on the name of the configuration and choose **Duplicate** or simply double-click the name of a read-only configuration.
- 2 Type a name for your new configuration. The “Sandbox” configuration is provided for you to test various checks against your project.
- 3 Check boxes next to validation checks you want this configuration to run. Checks are sorted into categories by standard and language. Any custom checks you have installed are listed in the Checks list. See *Writing CodeCheck Scripts* on page 255.



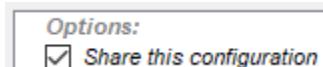
- 4 You can select any Dependency Checks you have configured for architecture nodes at the bottom of the list of checks. Click “Manage” to open the Configure Dependency Checks dialog. See page 203 for details.
- 5 When you select a check from the list, information about that check is shown below the list of checks. For some types of checks (such as metrics), you can change values to control how checks are performed. For example, the AC\_00 check lets you choose which types of control characters to disallow in source code files. The METRIC\_04 check lets you specify the maximum number of lines allowed in a function and choose whether to count only code lines or to include comment and blank lines.



- 6 To find checks for the types of violations you are looking for, click the search icon or press Ctrl+F and type a string in the search field. Use the arrows to move to the previous or next check that contains your search text in the short description of the check. The down arrow lets you limit the search to case sensitive matches or whole words.



- 7 If you want other users of this project to be able to use this CodeCheck configuration, check the **Share this configuration** option above the list of checks. Checking this box causes the configuration to be saved as part of the project. If the box is unchecked, the configuration is saved in a directory not available to other users

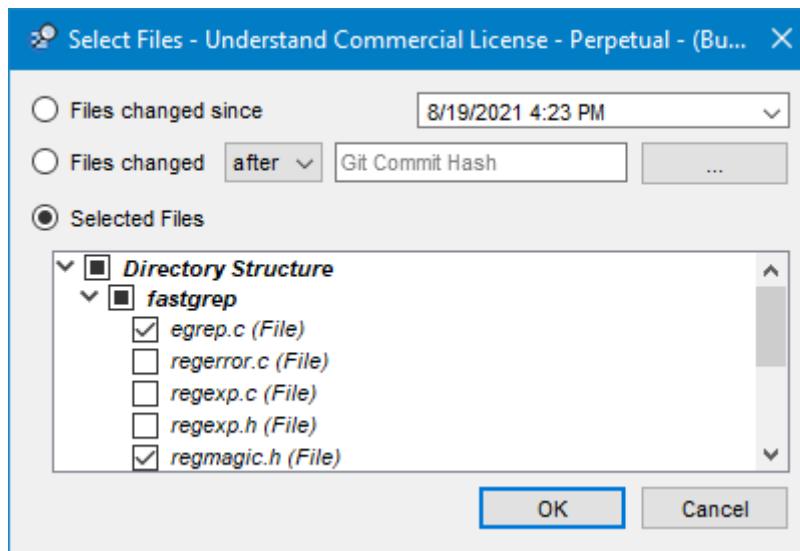


- 8 Click the save icon or press Ctrl+S to save changes to your configuration. Changes to a configuration are automatically saved when you select another configuration or close the dialog.
- 9 To resume editing a saved configuration, click the edit icon. You can also right-click on the name of the configuration and choose **Edit** or **Rename** or double-click the configuration name.
- 10 To delete a configuration, click the delete icon. You can also right-click on the name of the configuration and choose **Delete**.
- 11 To get a list of the checks performed by a configuration, right-click on the name of the configuration and choose **Copy All**. The list copied to your clipboard contains only the ID codes for each check, such as “MISRA12\_4.3”.

## Selecting Files to Check

Select which files to check using the **Files** drop-down list. You can choose any of the following:

- **All Files:** All files in the project are checked. If any source files have been edited but not yet saved, those edits are not checked; only the saved versions of files are checked.
- **Files changed since last run:** Files are checked if they have been changed and saved.
- **Uncommitted changes:** Files are checked if they are in the Git repository connected with the project and have changes that have been saved but not committed. If you have not yet specified a Git repository, see *History Options* on page 51.
- **Customize:** Opens the Select Files dialog to allow further control over which files are checked. The  filter icon opens this same dialog and can be used with the other file selection options in the drop-down.



The options in the Select Files dialog are:

- **Files changed since:** If no files have been changed since the date you select, the returns bar when you run the CodeCheck configuration shows that 0 Files were checked, and you will also see “Files (0)” by the Files drop-down list indicating that no files meet the criteria.
- **Files changed by Git Commit Hash:** Choose whether you want to check files changed after a certain commit or in a certain commit. Then paste the hash string into the field. If you have not yet specified a Git repository, see *History Options* on page 51. You can click the ... button to specify a Git repository.
- **Selected files:** Expand the architecture list as needed and check boxes next to architecture nodes to identify the files you want to check.

## Viewing Results

Use the **Results** drop-down to change how the results are displayed. After you perform a CodeCheck analysis, you can view the results in any of these ways without needing to re-run the checks:

- By Files (page 246)
- By Checks (page 247)
- Locator (page 248)
- Treemap (page 249)
- Log (page 250)

You can hide certain results by ignoring particular checks and violations or setting a baseline (page 250).

You can also print or export results (page 254).

### Viewing Results by File

By default, the results are listed by the file in which check violations occur. In the **Results** drop-down list, choose **By File** to return to this display.

The table lists the number of violations in each file and its full file path. Expand a file path to see a list of violations along with the entity affected, the line number, the column number, and the check ID. Expand the line for a violation to see a snippet of the code. You can double-click on the code to open the source file.

	Results	Entity	Line	Column	Check
6	Number of Results: 6				
1	▼ C:\Users\Yvonne\AppData\Roaming\SciTools\samples\fastgrep\fastgrep\regerror.c	s	5	6	CPP_-
	▼ ● Unused Parameter s in Non Virtual Functions				
	3 void				
	4 regerror(s)				
	5 char *s;				
	6 {				
	7 #ifdef ERRAVAIL				
2	> C:\Users\Yvonne\AppData\Roaming\SciTools\samples\fastgrep\fastgrep\regexp.h				
3	> C:\Users\Yvonne\AppData\Roaming\SciTools\samples\fastgrep\fastgrep\timer.c				

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.

Some commands in the CodeCheck hamburger menu allow you to show or hide various parts of this display. See *Running a CodeCheck* on page 242 and *Ignoring or Excluding Violations* on page 250 for details about other hamburger menu commands.

- **Show Violation Counts** toggles display of the number of violations.
- **Flatten Files List** toggles organizing files in a folder hierarchy that you can expand as needed or as a flat list of files as shown above.

## Viewing Results by Checks

In the **Results** drop-down list, choose **By Checks** to sort the results by the type of violation. This display is similar to the Results by File display (page 246). However, all violations of a particular type are listed together.

	Results	Entity	Line	Color
4	Number of Results: 4			
4	SciTools' Recommended Checks			
2	Overly Complex Functions - RECOMMENDED_10			
1	C:\Users\Yvonne\AppData\Roaming\SciTools\samples\fastgrep\fastgrep\egrep.c			
	Program overly complex (Complexity:{'CyclomaticModified': 30})	main	140	0
	138 char **args;			
	139			
	140 main(argc, argv)			
	141 int argc;			
	142 char *argv[];			
1	C:\Users\Yvonne\AppData\Roaming\SciTools\samples\fastgrep\fastgrep\regexp.c			
	Program overly complex (Complexity:{'CyclomaticModified': 22})	regmatch	803	0
2	Unused Local Variables - RECOMMENDED_14			
<b>Rationale</b>				
Overly complex programs are difficult to comprehend and have many possible paths making them difficult to test and validate. There are several variants for Cyclomatic Complexity:				

The organization under each violation is either a list of files if the **Flatten Files List** option in the CodeCheck hamburger menu is enabled or a folder hierarchy if the **Flatten Files List** option is disabled.

The **Show Violation Counts** option in the hamburger menu toggles the display of the number of violations in the left column.

If many violations of a particular type are detected, you might want to look at the individual checks in the **Checks** list 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. 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.

## Viewing Results in the Locator

In the **Results** drop-down list, choose **Locator** to display the results in a table that you can search using pattern matching and sorting based on the file name, violation name, line number, column number, and entity.

File	Violation	Line	Column	CheckID	Check Name	Entity
reg						
regerror.c	Unused Parameter	5	6	CPP_U003	A There shall...	s
regerror.c	Unused Parameter	5	6	MISRA08_0-1-11	0-1-11 A The...	s
regexp.c	Program overly ...	803	0	RECOMMENDED_10	Overly Compl...	regmatch
regexp.h	Unused tag decl...	8	15	CPP_U005	A project sho...	regexp
regexp.h	Unused tag decl...	8	15	MISRA12_2.4	2.4 A project ...	regexp

**Rationale**  
Unused function parameters are often due to design changes and can lead to mismatched parameter lists.

You can type values to match filenames and violations. Click the hamburger menu in a column header 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. To search for field values that do not contain a particular string, type ! (exclamation mark) and then the filter.

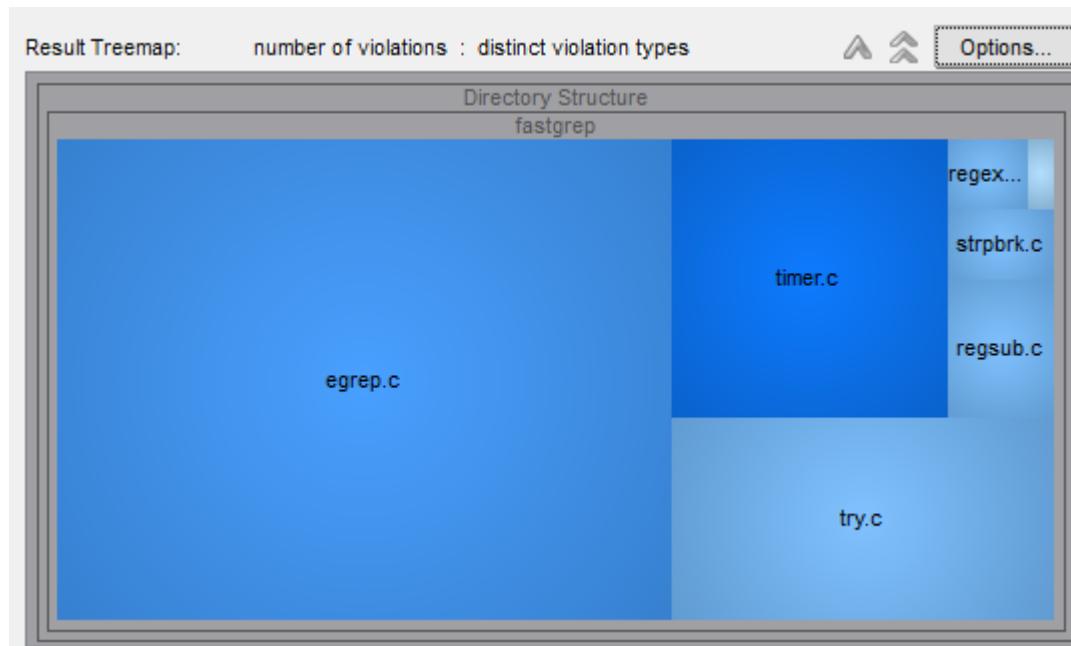
For details about using fields in the Locator, see *Filtering the List* on page 157.

A description of the selected check is shown below the results and five lines of code surrounding the violation are shown below the description. You can double-click a result to open the source file.

## Viewing Results by Treemap

In the **Results** drop-down list, choose **Treemap** to display the results as a 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.



You can double-click on a file block, to open that source file.

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 *Metrics Treemap* on page 213.

## Viewing the Results Log

In the **Results** drop-down list, choose **Log** to display the log created when the checks were run. The Log includes the number of files checked, how many checks were performed, and the number of violations found.



The screenshot shows a log window with the following content:

```

File selection set to Files changed since 2021-08-02
Completed 8/21/2021 12:21 AM

Selected Files:
Selected Checks:

Beginning Automatic Ignores Phase: Saturday, August 21, 2021 12:21:26 AM
Ending Automatic Ignores Phase: Saturday, August 21, 2021 12:21:26 AM
Begining File Inclusion Phase: Saturday, August 21, 2021 12:21:26 AM
Ending File Inclusion Phase: Saturday, August 21, 2021 12:21:26 AM
Begin Duration: Saturday, August 21, 2021 12:21:26 AM
Begin Inspection: Saturday, August 21, 2021 12:21:26 AM
Begin File Check Phase
File: fastgrep\regexp.c: Violations found
End File Check Phase
End Inspection: Saturday, August 21, 2021 12:21:27 AM
End Duration: 0 seconds

Inspection Summary:
Files: 2
Checks: 13
Violations Found: 41
Violations Ignored: 0
Violations Remaining: 41

```

If you enabled the **Use Verbose Logging** in the hamburger menu, the Log also includes a separate line for each violation found. This option must be selected before you click the **Inspect** button to take effect.

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.

## Ignoring or Excluding Violations

Violations may be excluded from the results either because they are ignored or because they are excluded:

- *Ignored violations* are manually added to the Ignores list by a user.
- *Excluded violations* are violations that are found but do not match the check criteria. For example, the Customize option lets you limit the results to violations in files changed since a specified date. Violations in files that have not been changed since that date are excluded violations.

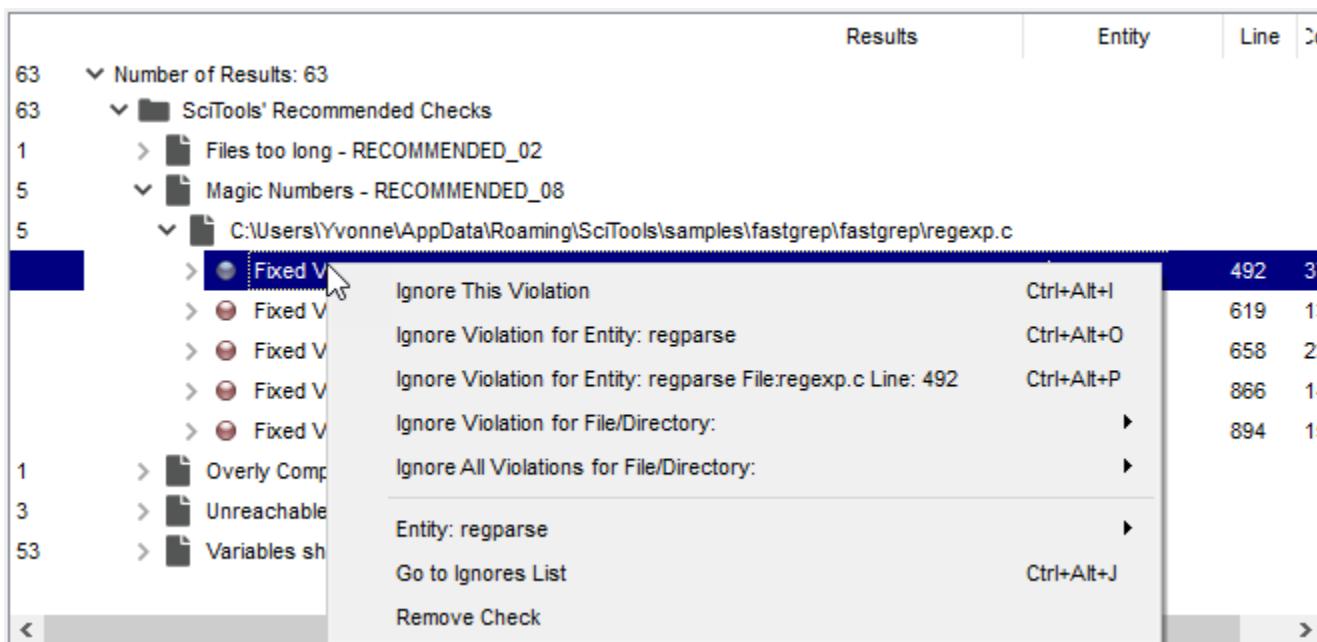
CodeCheck provides several ways to ignore violations in all or part of your project. For example, you might want to ignore violations in third-party code used by your project. Ignored violations can still be viewed in Understand, but are hidden by default.

If many violations are detected, you might want to look at the checks in your configuration 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 0 or 1 but other values are violations. For checks such as “Program Unit Cyclomatic Complexity”, you can set the maximum complexity that does not count as a violation.

## Ignoring Violations

Wherever you see a violation listed in the results, you can right-click and choose to ignore the violation. Several ignore settings are available:

- Ignore the violation instance only.
- Ignore this check violation for this entity wherever it occurs.
- Ignore this check violation for this entity only in this file and code line.
- Ignore violations of this check for the current file or for all files in a directory level you select above the location of this file. If you select a directory, violations in all of its subdirectories will be ignored.
- Ignore violations of all checks for the current file or for all files in a directory level you select above the location of this file. If you select a directory, violations in all of its subdirectories will be ignored.



If you decide that violations for a particular check need not be included in the results, choose **Remove Check**. This removes the violations from the current results, but not from the configuration. The next time you run the CodeCheck configuration, violations of this check will be shown again.

When you choose to ignore a violation, you are asked to confirm that it is OK to ignore multiple violations. Next, you are asked if you want to add a note about this ignored violation. If you click **Yes**, you can type text, for example to explain why it is ignored.

Violations that you choose to ignore are not listed in the Results list unless you enable **Show Ignored Violations** in the CodeCheck hamburger menu. They are highlighted if you show ignored violations and there is an X over the ball next to the violation description. Violation counts *do not* include ignored violations if violations are hidden but do include them if ignored violations are shown.

**Show Excluded Violations** toggles display of excluded violations. These are violations in files that were excluded from the results based on the file filtering.

If you have ignored any violations, click the **Ignores List** button to see a list of the currently ignored violations. You can search this list as you would the Result Locator list. See *Filtering the List* on page 157 for details.

To delete an ignored violation (that is, to restore it to the results list), select a row in the Ignores List view and click the  **Remove** icon in the CodeCheck toolbar. You can also remove a selected ignore using the right-click menu or by pressing the Delete key.

See *Running a CodeCheck* on page 242 and *Viewing Results* on page 246 for details about other hamburger menu commands.

---

### Adding Notes About Ignored Violations

You can add notes about each row in the Ignores List and limit each ignore to certain lines in the file. Notes are shown in the Note column and line specifications are shown in the Lines column. You can sort and filter based on these columns.

- **Note:** This is text information such as a comment about an ignore. To add a note for an ignore, select a row and click the  **Add/Edit Note** icon. Type a note and click OK. To remove a note, select a row and click the  **Remove Note Text** icon. You can also add and remove notes using the right-click menu or by pressing the + and - keys.
- **Lines:** This is a line number or range of line numbers where the ignore applies. To specify lines for an ignore, right click on a row and choose **Add/Edit Lines**. For example, type 65 to apply this ignore only on line 65; type 60-70 to apply the ignore to that range of lines; type \* to apply the ignore to all lines in the file. To remove the line specification, open the same dialog and remove the values.

## Using Baseline Ignore Settings

CodeCheck lets you set a baseline to ignore existing violations in older code in order to identify violations in new or modified code.

In the Results and Ignores lists, baseline ignores have a green background, while regular violation ignores have a blue background. Ignores created with comments in code (see *Adding Automatic Ignores to Code* on page 253) have an orange background. (The selected row has a dark blue background no matter what type of ignore or result it is.)

File	Entity	Violation	CheckID	Lines
C:\Users\Yvo...	regmatch	Program overly complex ...	RECOMMENDED_10	*
C:\Users\Yvo...	regexp	Unused tag declaration %1	CPP_U005	*
C:\Users\Yvo...	*	*	*	4
C:\Users\Yvo...	*	*	MISRA12_8.3	18

To set all current violations as baseline violations:

- 1 Run a CodeCheck configuration.
- 2 Click the  **Set all current violations as baseline** icon in the CodeCheck toolbar.
- 3 Click **OK** to allow multiple baseline ignores to be created.
- 4 Click the **Ignores List** button to see the list of baseline ignores that were created. (Baseline ignores are hidden from this list if **Show Baseline Ignores** is toggled off in the CodeCheck  hamburger menu.)

To change a baseline ignore to a simple ignore, right-click on the row and choose **Set as manual in Ignores list**.

To change a simple ignore to a baseline ignore, right-click on the row and choose **Set as a Baseline in Ignores list**.

To remove a baseline ignore from the Ignores List entirely, right-click on the row and choose **Remove from Ignores list**.

To remove *all* baseline ignores from the Ignores List entirely, click the  **Remove all Baseline False Positives** icon in the CodeCheck toolbar.

To hide or show baseline violations in the Results lists (by file, by check, etc), toggle **Show Baseline Violations** in the CodeCheck  hamburger menu. Note that if you enable showing baseline violations, all ignored violations are shown because baseline ignores are a type of ignore.

## Adding Automatic Ignores to Code

In addition, you can add comments to your code to cause *Understand* to ignore specific violations. The general format is as follows:

```
statement; //UndCC_Line(*) Note about approval of violation
```

For example:

```
goto FOO; //UndCC_Line(*) Use of goto OKed by Gerry on Nov 7, 2019
```

Such comments create a rule in the Ignores List to ignore all CodeCheck violations on this line and add a note about the approval process. Either C style (`/* */`) or C++ style (`//`) comments may be used.

The following additional keywords let you create ignore rules for the next line, a range of lines, an entire file, an entity, or a range of lines within an entity.

- `//UndCC_NextLine(*)` Ignore all violations on the next line.
- `//UndCC_Begin(*)` Ignore all violations from the start of this line until `_End`.
- `//UndCC_End(*)`
- `//UndCC_File(*)` Ignore all violations in this file.
- `//UndCC_Entity(EntityName,*)` For the next entity named EntityName, ignore all violations anywhere.
- `//UndCC_EntityBegin(EntityName,*)` For the next entity named EntityName, ignore violations until `_EntityEnd`.
- `//UndCC_EntityEnd(EntityName,*)`

The asterisk after the keyword can be replaced with a CheckID or list of CheckIDs so that only the specified violations are ignored. For example, the following example ignores only violations of MISRA rule 2.1:

```
stmt; //UndCC_File(MISRA12_2.1) Code reachable by external app
```

For additional details and examples, see the “CodeCheck Comment Keywords” topic on the [support website](#).

You can toggle **Require Explicit CheckID on Automatic Ignores** on in the CodeCheck  hamburger menu to require that all `//UndCC` comments identify a specific CheckID instead of using an asterisk. If `*` is used in such comments, the comment is ignored.

---

## Exporting CodeCheck Results

CodeCheck provides a number of ways to use the results.

- **Printing:** You can print results from the Files, Checks, Locator, and Treemap views to a PDF file or a physical printer by clicking the  **Print** icon in the CodeCheck toolbar. See *Graphical View Printing* on page 238.
- **Exporting:** You can export results from the Files, Checks, and Locator views to HTML, your clipboard, or a text file by clicking the  **Export** icon in the CodeCheck toolbar. The exported text is a tab-separated values file that lists the full file path, the internal ID string used to identify the entity, the type of violation that

occurred, and the check being performed that identified this violation. The HTML output also includes columns for whether the violation is ignored and any notes about ignored violations.

In the Treemap view, you can click the  **Export** icon to save the treemap to a PNG, JPG, or SVG file.

The Log view can be copied to the clipboard using the  **Copy** icon or exported to a text file using the  **Export** icon.

In the Results by Check view, the  **Export** icon allows you to export either detailed or summary results. The summary report is similar to the following:

```
CodeCheck Summary
Number of Results: 704
SciTools' Recommended Checks704
    """Commented Out"" Code7
    Definitions in Header Files28
    Functions Too Long5
    Goto Statements28
    Magic Numbers300
    Overly Complex Functions10
    Unused Functions36
    Unused Local Variables50
    Variables should be commented15
```

## Writing CodeCheck Scripts

CodeCheck scripts are special Python or Perl scripts that let you create 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 Python or Perl API along with a set of special functions designed to interact with the Understand CodeCheck interface. The following examples use the Perl API. See *APIs* on page 284 and the API/Plugins category on the [support website](#) for information about the Python API.

You can also use CodeCheck from the "und" command line. For details, see *Using CodeCheck* on page 294.

CodeCheck script files have a \*.upl extension for scripts written in Perl and \*.upy for scripts written in Python.

To begin writing your own check, follow these steps:

**1** Find template scripts to modify in the appropriate location for your operating system:

- **Windows:** C:\Program Files\SciTools\conf\plugin\User\CodeCheck or  
C:\Users\<userID>\AppData\Roaming\SciTools\plugin\Codecheck
- **MacOS:** ~/Library/Application Support/SciTools/plugin/User/Custom/  
(Note: You will need to create the plugin/User/Codecheck folder.)

- **Linux:** <install\_path>/SciTools/conf/plugin/User/Codecheck or  
~/.config/SciTools/plugin/Codecheck (Note: You will need to create the  
plugin/Codecheck directory.)

- 2 Edit the template file using a text editor.
- 3 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 unique identifier for this check.
sub checkID { return "TEST_01_CHAR_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."; }
```

- 4 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;
}
```

- 5 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; }
```

- 6 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; }
```

- 7 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++;
    }
}
```

- 8** 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: `perl -c mysample.upl`.

To learn more, see the API/Plugins category on the [support website](#). Browsing the CodeCheck scripts that are shipped with Understand can also be very beneficial. If you have questions about CodeCheck scripts, the CodeCheck category on the [support website](#) can be a great place to learn more.

## Installing Custom Scripts

You can install a script 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.

---

## Chapter 12 Comparing Source Code

This chapter explains the history and source-code comparison features provided by *Understand*.

This chapter contains the following sections:

---

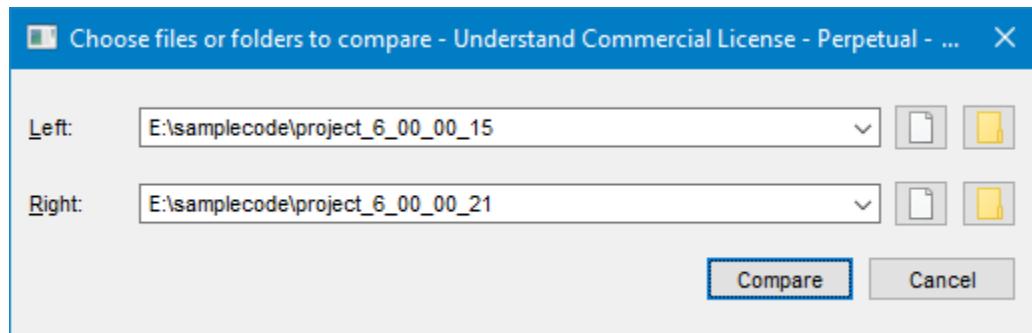
Section	Page
Comparing Files and Folders	259
Comparing Entities	262
Comparing Text	263
Comparing Projects	263
Comparison Graphs	266
Exploring Git History	267
Exploring Differences	269

---

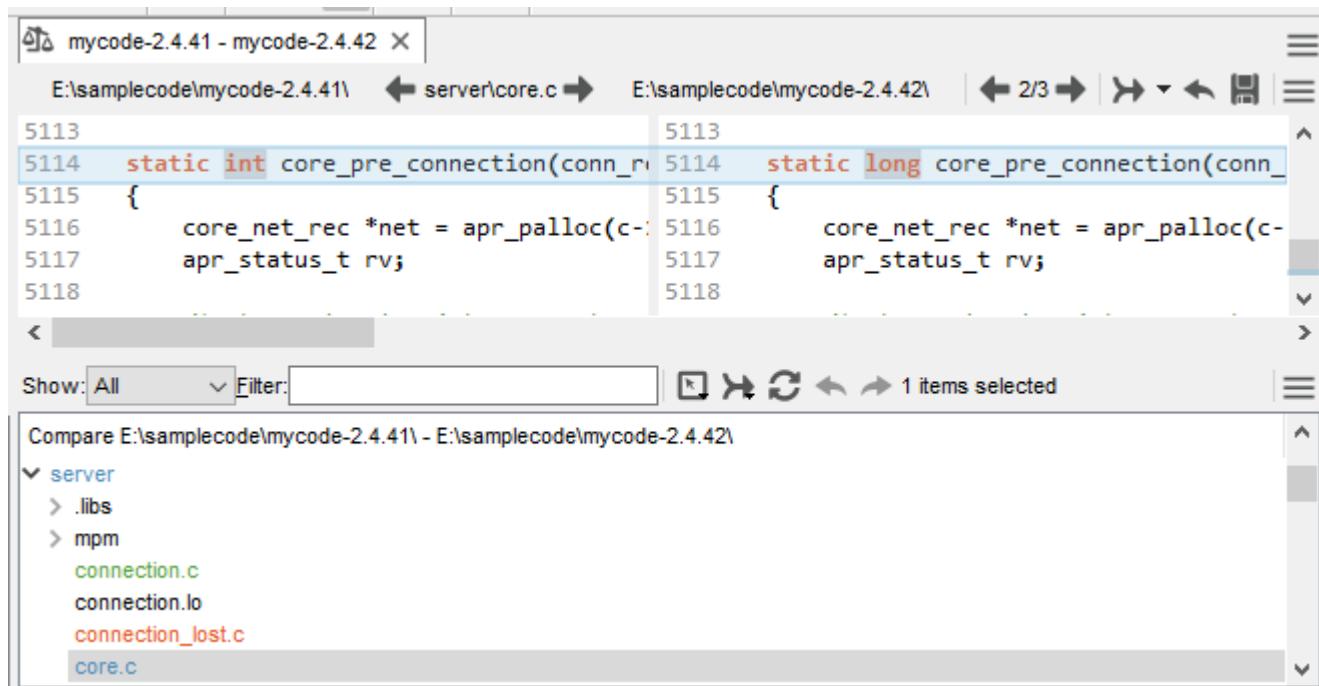
## Comparing Files and Folders

*Understand* provides a tool for comparing files and folders. You can use this tool to compare files or directory trees that are similar or to compare older and newer copies of a file or directory tree. To compare entities in older and newer versions of an *Understand* project, you can also use the Changed Entities view, which is described on page 263.

To open this tool, choose **Compare > Compare Files/Folders** from the menus.



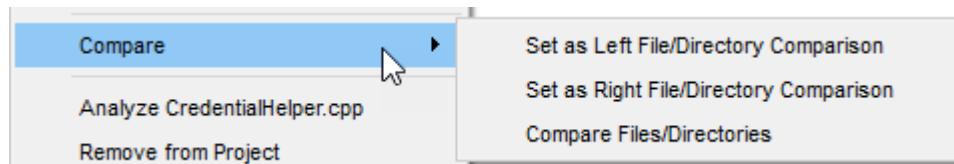
In this dialog, select a file or folder for both the left and right comparison. Both sides should be similar files or similar folders. Click the file button to browse for a file; click the folder button to browse for a directory. Then click **Compare**.



Subdirectories of the directories you choose are also compared.

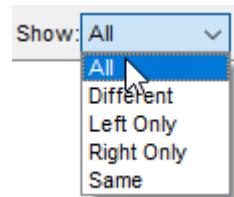
The code comparison section of the results window is described on page 269. If you are comparing folders, there is an additional bottom pane that shows the folder-level and file-level differences and lets you filter the contents you want to compare.

A quick way to compare two files in a project is to right-click on the name of one file (for example, in the Project Browser) and choose **Compare > Set as Left File/Directory Comparison** from the context menu, right-click another files and choose **Compare > Set as Right File/Directory Comparison**, and then choose **Compare > Compare Files/Directories**.



The names of changed files and folders are shown in blue. Files and folders that exist only in the right version (added) are shown in green, and files and folders that exist only in the left version (deleted) are shown in red. You can choose **Show Icons** from the hamburger menu to include folder and file icons in the list. These icons have arrows and the ≠ sign to indicate whether the contents have been added, deleted, or changed.

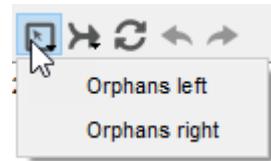
By default, all files and folders are listed. You can use the **Show** drop-down to choose whether to restrict the list to showing only:



- **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 (deleted). 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 (added). 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, typing `sim` matches any folders or files with “sim” in their names. All files within folders that match the **Filter** (and the **Show** drop-down setting) are shown. Filtering occurs only after you click the Rescan icon. If you want to use regular expressions, enable that option in the hamburger menu . Wildcards are not recognized.

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 icon 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**. The number of items selected is shown in the toolbar.



You can copy folders and files from one side to the other. 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 icon and choose either **to the left** or **to the right**. This opens

the Copy Files dialog, which lists the files or folders to be copied. If the list is correct, click **OK**. Click the icon to undo your last copy/merge change. Click the icon to redo your last undo.

If you have modified a file on the right, you can click the Save icon to save that file to its existing location. If you modify a file and then switch to the comparison of another file, you are asked whether you want to save and recompare the files.

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. However, there is no way to save the file on the left

CodeChecked: Never | Line: 21 Column: 17 | Tab Width: 8 | RW | Fortran

The code comparison section of the results window is described in more detail on page 269.

## Comparing Entities

You can compare two entities by choosing **Compare > Compare Entities** from the menus. You see the Compare Entities window. Select entities in both the left and right entity filter panes to see a comparison.

A quick way to compare the code that defines two entities in a project is to right-click on the name of one entity and choose **Compare > Set as Left Entity Comparison** from the context menu, right-click another files and choose **Compare > Set as Right Entity Comparison**, and then choose **Compare > Compare Entities**.

The screenshot shows the 'Compare Entities' window with two code snippets side-by-side. The left snippet is for 'decode' and the right snippet is also for 'decode'. Both snippets are identical. Below the code are two entity filter panes. The left pane contains 'crc32\_little', 'decode', 'decode', and 'decomm'. The right pane also contains 'crc32\_little', 'decode', 'decode', and 'decomm'. A 'Show' dropdown menu is set to 'C++ Functions'. A 'Filter:' input field is present at the bottom of each pane.

```

33     left = (MAXBITS+1) - len;
34     if (left == 0) break;
35     if (s->left == 0) {
36         s->left = s->infun(s->inhow,
37         if (s->left == 0) longjmp(s-
38     }
39     bitbuf = *(s->in)++;
40     s->left--;
41     if (left > 8) left = 8;
42 }
43 return -9;
44 }
```

The top portion is similar to other types of comparisons (page 269).

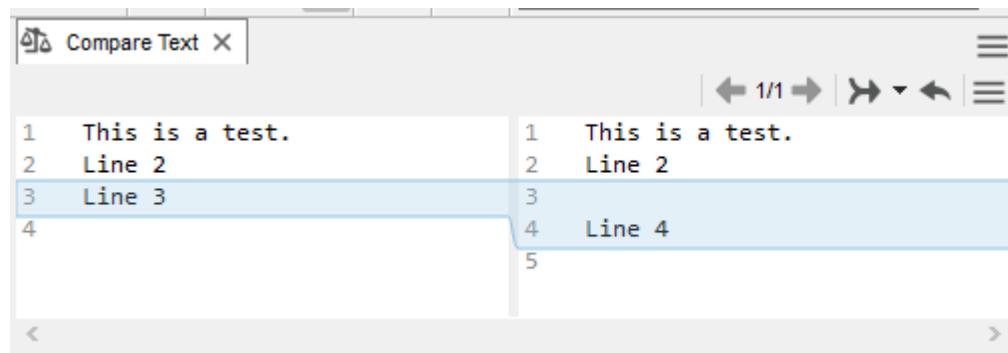
Below the comparison is an entity filter (page 124). Select a type of entity in the **Show** drop-down. Then use the lists to select two entities you want to compare. The **Filter** fields let you type a string you want to match anywhere in the entity name. Filtering occurs as you type. Wildcards and 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.

If you are comparing a file, you can click the Merge icon to merge changes into the file on the right and click the Save icon to save changes to the file on the right.

## Comparing Text

You can compare text that you paste into a window by choosing **Compare > Compare Arbitrary Text** from the menus. Paste before and after text to compare into the left and right sides. The two sides are compared automatically.:



The text comparison is similar to the comparison between two entities. You can merge and unmerge differences, but cannot save files. Copy and paste into another file window if you need to save the merged text.

A quick way to compare text is to highlight some text, right-click, and choose **Compare > Set as Left-Text Comparison** from the context menu. Then highlight other text, right-click, and choose **Compare > Set as Right-Text Comparison**. Finally, choose **Compare > Compare Text Selections**.

## Comparing Projects

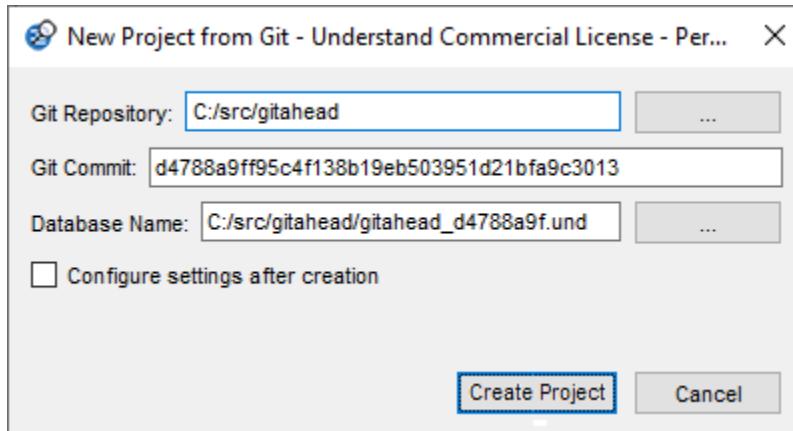
You can compare older and newer (or variant) versions of the same project if you have complete, separate copies of both versions of the project or if you are using Git to manage your source code.

To compare different versions of a project, follow these steps:

- 1 Open the newer version of the project.
- 2 Choose **Compare > Compare Projects** from the menus. This opens the Comparison Projects area.



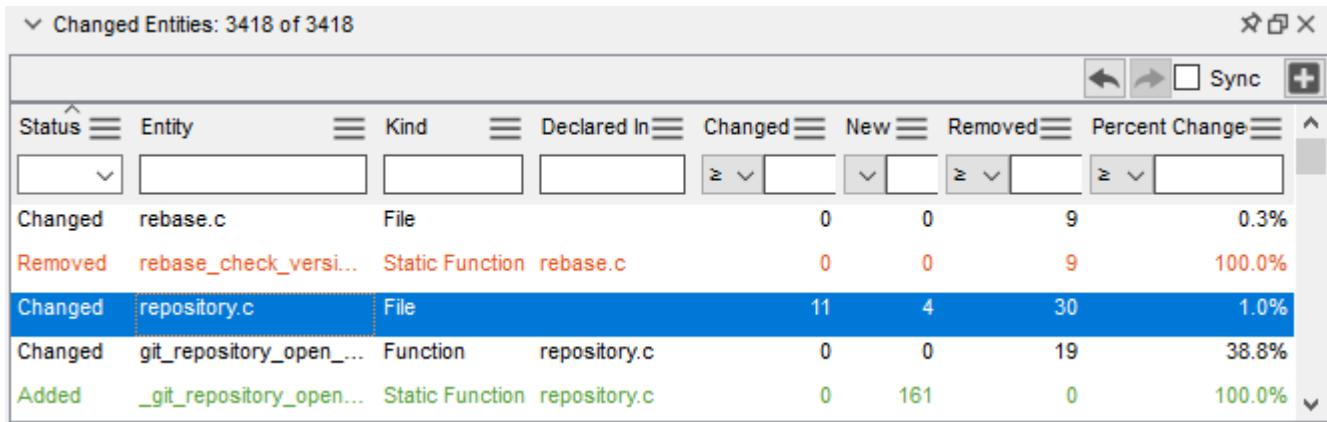
- 3 Select a previous version of the project in one of the following ways:
  - **Git revision:** Click the  **Create from Git** icon. In the New Project from Git dialog, browse to the location of the top-level directory that was cloned from the **Git Repository**, paste the **Git Commit** hash string for the older version of the project you want to compare to the current version, and provide a directory location and **Database Name** where you want to create an *Understand* project that matches this older Git commit. To make the project settings of the created project match those of your current project, check the **Configure settings after creation** box and see page 39. Click **Create Project**.



- **Older or variant project database:** Click the  **Use Existing Database** icon. Browse to select another copy of the \*.und directory where an older or variant version of this *Understand* project is stored. Click **OK**.
- 4 You may be prompted to analyze the new or existing database. Click **OK** and wait for the analysis to complete. If you want to force analysis of a comparison project, click the  **Analyze** icon in the Comparison Projects area.
  - 5 If multiple projects are listed in the Comparison Projects area, select the one you want to compare to the currently open project.
  - 6 Choose **Compare > Locate Change Entities** from the menus or click **Locate Change Entities** at the bottom of the Comparison Projects area.

This opens the Changed Entities Locator area at the bottom of your *Understand* window and comparison panes at the top. Use of the comparison panes is described in *Exploring Differences* on page 269. Use of the Changed Entities Locator is similar to use of the *Entity Locator* on page 155.
  - 7 Select an entity from the list of changed entities so that you can view the changes.
  - 8 You can merge changes from the previous version of the project to the current version by clicking the  **Merge** icon. Click the  **Save** icon to save changes to the version of the file on the right.

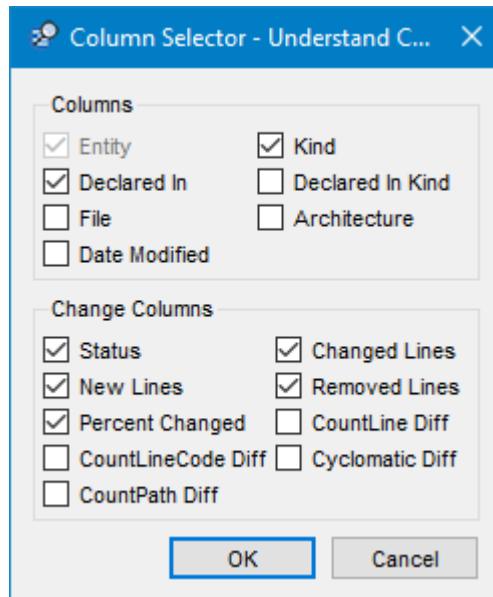
By default, the columns in the Changed Entities Locator are Status (changed, added, or removed), Entity, Kind, Declared In, Changed Lines, New Lines, Removed Lines, and Percent Changed.



The screenshot shows a software interface titled 'Changed Entities: 3418 of 3418'. The main area is a grid table with the following columns: Status, Entity, Kind, Declared In, Changed, New, Removed, and Percent Change. The rows represent different entities and their metrics. The last row, 'Added \_git\_repository\_open...', has a blue background, indicating it is selected.

Status	Entity	Kind	Declared In	Changed	New	Removed	Percent Change
Changed	rebase.c	File		0	0	9	0.3%
Removed	rebase_check_versi...	Static Function	rebase.c	0	0	9	100.0%
Changed	repository.c	File		11	4	30	1.0%
Changed	git_repository_open_...	Function	repository.c	0	0	19	38.8%
Added	_git_repository_open...	Static Function	repository.c	0	161	0	100.0%

You can click the plus icon in the toolbar to add columns such as the architecture path, the date the file was modified, and the differences between old and new values for several metrics. You can sort and filter on all of these columns.



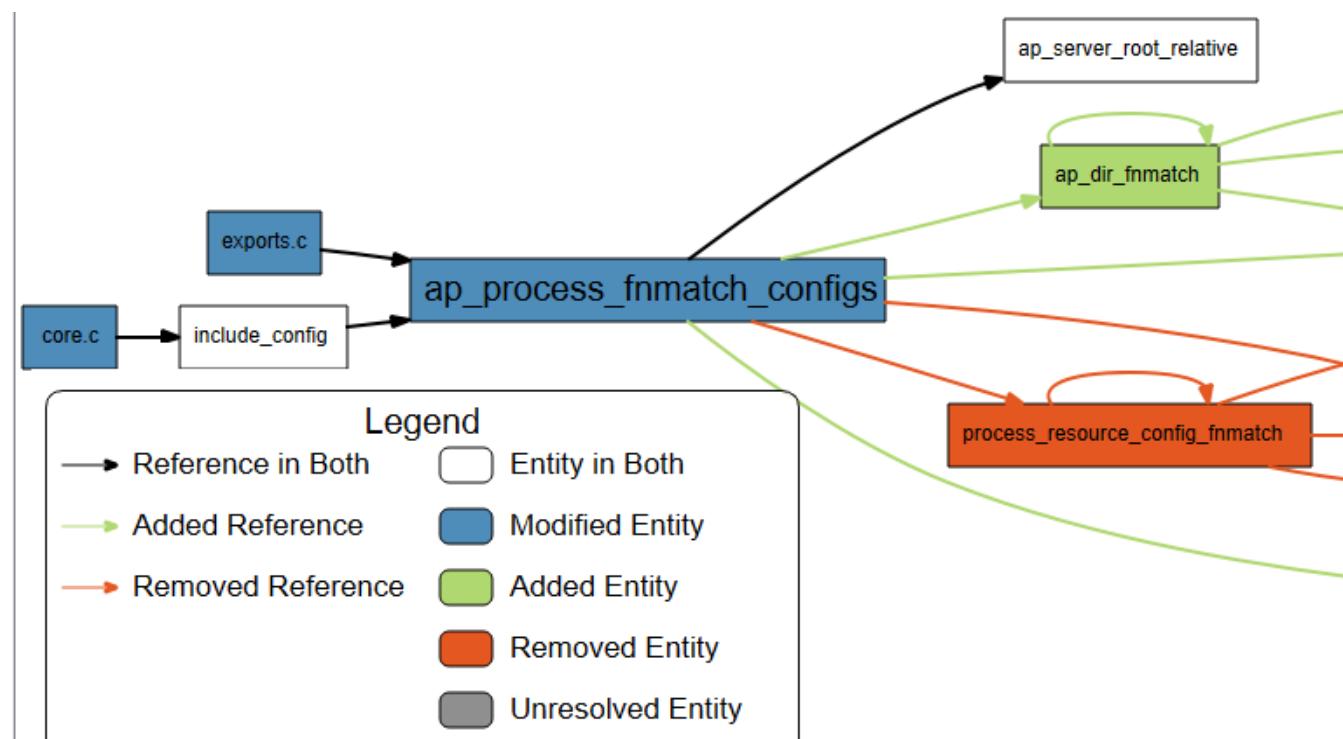
Once you have set a comparison project, you can display the Compare Butterfly and Compare Control Flow graphs (page 266).

You can close the previous project (to save memory) by clicking the  **Close Comparison Project** icon.

## Comparison Graphs

You can create graphs to compare older and newer versions of the same project if you have complete, separate copies of both versions of the project. To create such graphs, follow these steps:

- 1 Specify a comparison project as described in *Comparing Projects* on page 263.
- 2 Right click on an entity, such as a function, and choose a comparison graph from **Graphical Views**. Depending on the type of entity you select, the comparison graphs may include **Compare Control Flow**, **Compare Butterfly**, or **Compare Object References**.
- 3 Comparison graphs highlight modified, added, removed, and unresolved entities and references using the colors shown in the Legend. You can change these colors in the **Graphs** category of the **Tools > Options** dialog (page 114).



- 4 Click any node in the graph to jump to the changes made between the two versions of the project in the file differences panes below the graph.

## Exploring Git History

*Understand* integrates with Git if you use it to for source code control. You can view the commit history and other Git information directly in your source code. This includes seeing who has changed or created a file, viewing version differences side-by-side, and seeing commit information, all without leaving *Understand*.

*Understand* automatically finds the Git information for most functionality. The exception is when filtering by Uncommitted Changes in CodeCheck. To use this feature, specify the Git repository in the History category of the Project Configuration (page 51).

If you will be using Git integration in the Source Editor windows, open the Options dialog by choosing **Tools > Options** (or **Understand > Preferences** on MacOS) and toggling on the following options:

- In the **Editor** category, toggle on the **Git** option in the Toolbars area.
- In the **Editor** category, toggle on the **Blame** option in the Margins area.
- In the **Editor > Advanced** category, toggle on the **Show Inline Blame** option in the Version Control area, which you need to scroll down to see.

When the blame margin is enabled, along the left margin of the Editor you'll see the initials of the programmer that wrote or changed each line of code. If you point to the initials, you see the author, commit date, commit message, and commit ID as hover text.

```

30 BD
31 |     #include "gmock/gmock.h"
32 |     #include "gmock/internal/gmock-port.h"
33 |
34 BD ▼ namespace testing {
35 |
36 |     ▼ GMOCK_DEFINE_bool_(catch_leakedMocks, true,
37 KK |             "true if and only if Google Mock should report leaked "
38 |             "mock objects as failures.");
39 B Krys Kurek August 12, 2019 7:09am
40 | restore mistakenly removed iiffs in their explicit form ::kWarningVerbosity,
41 |                                         "bose Google Mock's output is."
42 | Due to confusion arisen from "iff" standing for "if and n"
43 | only if", ts all messages.\n",
44 | this commit uses the latter.    " warning - prints warnings and errors.\n"
45 |                                     " error - prints errors only.");
46 |
47 A ▼ GMOCK_DEFINE_int32_(defaultMockBehavior, 1,
48 | 7bd4a7f3                                         "Controls the default behavior of mocks."
49 |                                         " Valid values:\n"

```

You can toggle the blame margin on and off using the  icon in the Source Editor toolbar.

If you modify a file with the blame margin enabled, change bars are added as you edit to indicate uncommitted changes.

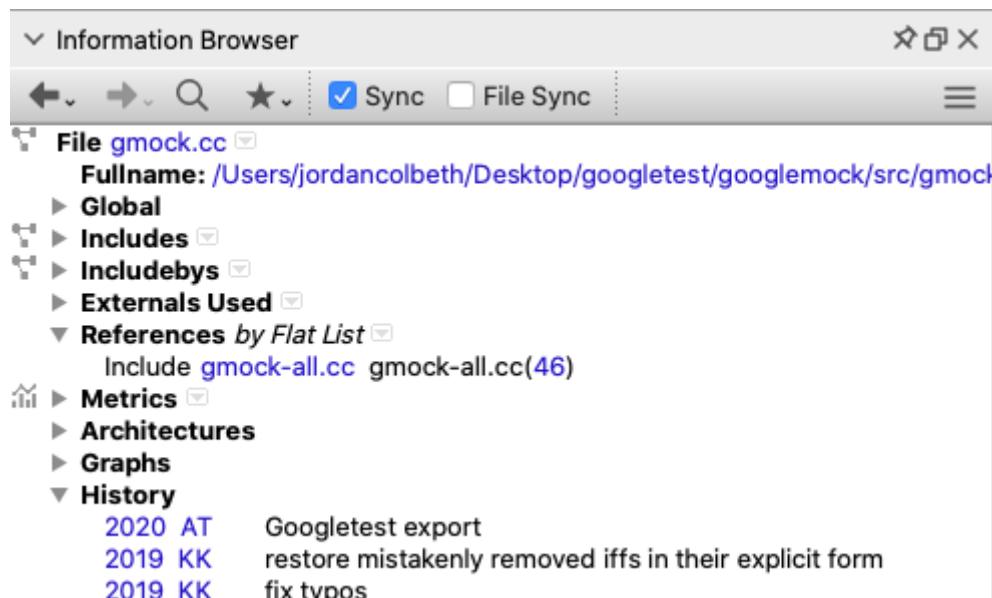
When inline blame information is enabled, commit information is shown inline to the right of code lines. Only Git commit information for the currently selected line is shown. Hover your mouse over the line to see more of the commit message if it is long.

```

154 | // Do we see a Mock flag?
155 | ▾ if (MockBoolFlag(arg, "catch_leakedMocks",
156 |           &MOCK_FLAG(catch_leakedMocks)) ||
157 AW  MockStrFlag(arg, "verbose", &MOCK_FLAG(verbose)) || Aly Wik, Aug 2017: Adding flag
158 |     MockIntFlag(arg, "default_mock_behavior",
159 |                   &MOCK_FLAG(default_mock_behavior))) {
160 BD  // Yes. Shift the remainder of the argv list left by one. Note
161 |     // that argv has (*argc + 1) elements, the last one always being

```

In addition, the Information Browser shows any relevant Git information for a file in the **History** node. Expand the History drop-down to see a chronological list of every commit made to the file.



You can open a Commit History view that is similar to other comparison views by:

- Clicking a commit indicator in the blame margin of a Source Editor window.
- Clicking a commit in the History list in the Information Browser.
- Clicking the icon in the Source Editor toolbar to compare the last committed version to your current uncommitted changes.

In this view you can compare changes that were made in any commit side-by-side with the previous version of the file. The newer version is shown in the right pane, and the immediately previous version of the file is shown in the left pane. The toolbar of this tab shows the dates both versions were created or changed, and you can cycle between earlier and later commits by clicking the arrow icons around the file name in the toolbar. When the right pane is showing your uncommitted working version, you can use the Revert icon in the toolbar to undo the currently selected difference between the last

committed version and your working version. If multiple differences are found, you can navigate to the next or previous difference using the  2/3  arrows around the number of differences.

Aug 14, 2018 3:45pm ← gmock.cc → Oct 04, 2018 6:28pm ← 1/6 → ⌂ ⌃ ⌁

```
56 |  
62 BD namespace internal {  
63  
64     static const char* ParseGoogleMockFlagVal  
65  
66  
67     // str and flag must not be NULL.  
68     if (str == NULL || flag == NULL) return  
69  
70     // The flag must start with "--gmock_".  
71     const std::string flag_str = std::strir
```

```
56 |  
57     namespace internal {  
58  
59     static const char* ParseGoogleMockFlagVal  
60  
61  
62     // str and flag must not be NULL.  
63     if (str == nullptr || flag == nullptr) return  
64  
65     // The flag must start with "--gmock_".  
66     const std::string flag_str = std::strir
```

## ***Exploring Differences***

When you compare items, you see a comparison window. The area below the comparison differs depending on what you are comparing. The controls for the comparison itself remain similar across various types of comparisons.

A screenshot of a terminal window titled "mycode-2.4.41 - mycode-2.4.42". The window shows two panes: the left pane is at "E:\samplecode\mycode-2.4.41\" and the right pane is at "E:\samplecode\mycode-2.4.42\". Both panes show the number "5108". The status bar at the bottom indicates "2/3" and has navigation icons.

The left pane shows the first item you are comparing; the right pane shows the second item. The names or paths for the items being compared are shown above the two comparison panes.

If you are comparing multiple folders, you can move to the next or previous file using the  `servercore.c`  arrows around the filepath.

If multiple differences are found, you can navigate to the next or previous difference using the   arrows around the number of differences.

Right-clicking in either source pane displays a context menu similar to the one in the Source Editor. See *Context Menu* on page 170 for details.

Scrolling the two panes is synchronized horizontally and vertically. The vertical scrollbar shows the location and size of changed sections of code using the highlight color.

When comparing files, the file on the left is in read-only mode by default, 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. However, there is no way to save the file on the left

CodeChecked: Never | Line: 21 Column: 17 | Tab Width: 8 | RW | Fortran

When comparing files, you can merge the currently selected difference in the version on the left into the file on the right by clicking the Merge icon. Click the icon to undo your last merge change. To merge all differences from the left file to the right file, use the drop-down arrow next to the Merge icon to select **Merge All**. Merging differences is not supported when comparing entities.

If you have modified a file on the right, you can click the Save icon to save that file to its existing location. If you modify a file and then switch to the comparison of another file, you are asked whether you want to save and recompare the files.

The hamburger menu for the comparison provides the following commands:

- **Case Insensitive:** By default, changing the case of a letter is not treated as a difference. For example, if you change “a” to “A”, no difference is highlighted. You can change this behavior by toggling this option off.
- **Skip Whitespace:** By default, changing the number of spaces or tabs is not treated as a difference. No difference is found if only whitespace was changed. You can change this behavior by toggling this option off.
- **Files are Unicode:** By default, differences are reported only for ASCII files. If *Understand* says “File is Binary”, toggle this option to turn on Unicode file handling.
- **Highlight Different Words:** By default, the entire line that contains a difference is highlighted. Toggle this option to also highlight individual words that are different in another color.
- **Different Word Color:** Choose the color to use to highlight words, numbers, or symbols that are different in the comparisons. This color is an overlay that is combined with the other highlight color as appropriate.
- **Highlight Color:** Choose the color to use to highlight lines that are different in the comparison.
- **Release Window:** Toggle this option to open the comparison in its own separate window. This is useful when comparing larger files.

The Case Insensitive, Skip Whitespace, and Files are Unicode options are not available if you have made a change to a file.

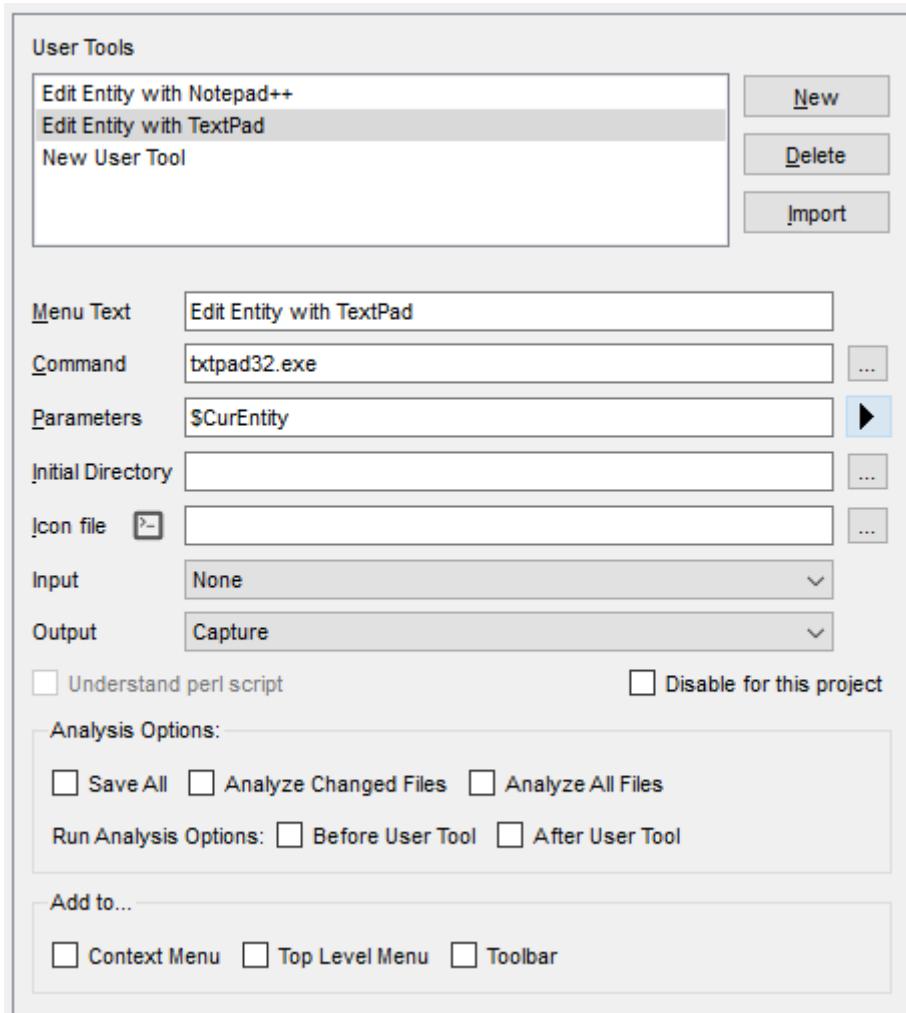
This chapter will show you how to configure and use source code editors and other external tools from within *Understand*.

This chapter contains the following sections:

<b>Section</b>	<b>Page</b>
Configuring Tools	272
Adding Tools to the Context Menus	279
Adding Tools to the Tools Menu	280
Adding Tools to the Toolbar	281
Importing and Exporting Tool Commands	282
Running External Commands	283
Running Plugins: Graphs and Interactive Reports	284

## Configuring Tools

Select **Tools > User Tools > Configure** from the menus to open the User Tools category of the Options 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 Options dialog provides sub-categories for the User Tools category to control how the tools you configure may be launched.



First, use the **User Tools** category of the Options 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 274 for a full list of variables.

- 3 If the tool you use is on your executable search path, simply type its filename 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 274 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. You can use the < sign to separate parameters that need to come from stdin. For example, if the password for a tool needs to come from stdin, use: < \$PromptForPassword

Quote marks in the parameter list are handled the same as quotes in the Microsoft Windows command prompt window and a Linux terminal. This is a change from previous behavior. For example:

Parameter Text	Old Result	New Result
\"some text\"	2 args = { \ , some text\ }	2 args = { "some, text" }
-arg="a b"	2 args = { -arg= , a b }	1 arg = { -arg=a b }
-arg=\$Prompt...	2 args = { -arg=, result }	1 arg = { -arg=result }

- 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 Check the **Disable for this project** box if you do not want this user tool to be available when the current project is open.
- 11 In the “Analysis Options” area, choose actions you would like to be performed before and/or after this user tool has completed its action and returned. These

actions can include saving all files, re-scanning for new files in project directories, analyzing modified files, and analyzing all files.

- 12** In the “Add to...” area, choose ways you want to access this command in *Understand*. The **Context Menu** checkbox adds the tool to the right-click context menu. The **Top Level 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 the **Delete** button.

For information about using the **Import** button, see *Importing and Exporting Tool Commands* on page 282.

---

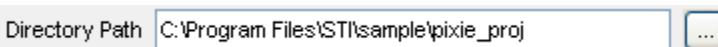
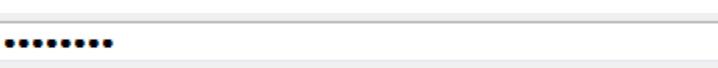
## Variables

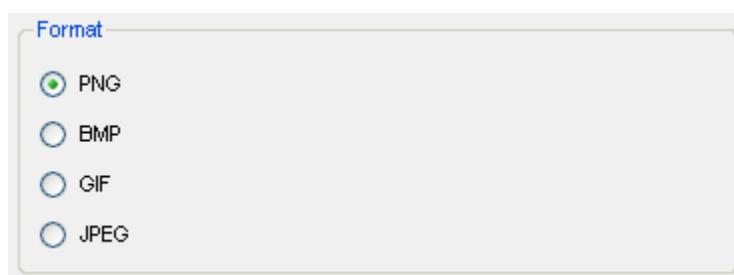
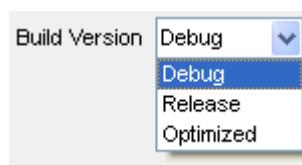
Variables beginning with \$Cur are current position variables that apply only from a Source Editor window. Variables beginning with \$Decl are declaration variables that apply only when an entity with a declaration is selected. Variables beginning with \$Prompt display a dialog to ask the user for some information.

You can use the following variables in the Command or the Parameter field.

Variable	Description
\$CppIncludes	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.
\$CppMacros	Lists all of the macro definitions specified in the Project Configuration.
\$CurArchitecture	Name of current architecture.
\$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.

Variable	Description
\$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.
\$DeclFileNameShortName	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.
\$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.
\$NoFileLink	Displays file names as text only and not as links.
\$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, \$PromptForCheckBox "Show Debug Text" displays the following prompt
	<input type="checkbox"/> Show Debug Text
\$PromptForCheckBoxGH	Prompts user with a series of checkboxes displayed in a horizontal group. For example, \$PromptForCheckBoxGH "Show=Debug Text;Tool Tips;Line Numbers" 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.
	<input type="checkbox"/> Show <input type="checkbox"/> Debug Text <input type="checkbox"/> Tool Tips <input type="checkbox"/> Line Numbers

Variable	Description
\$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.
	
\$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:
	
\$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:
	
\$PromptForPassword	Prompts user for a password. Characters typed in this field are obscured.
	
\$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.
	

Variable	Description
\$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.
	
\$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.

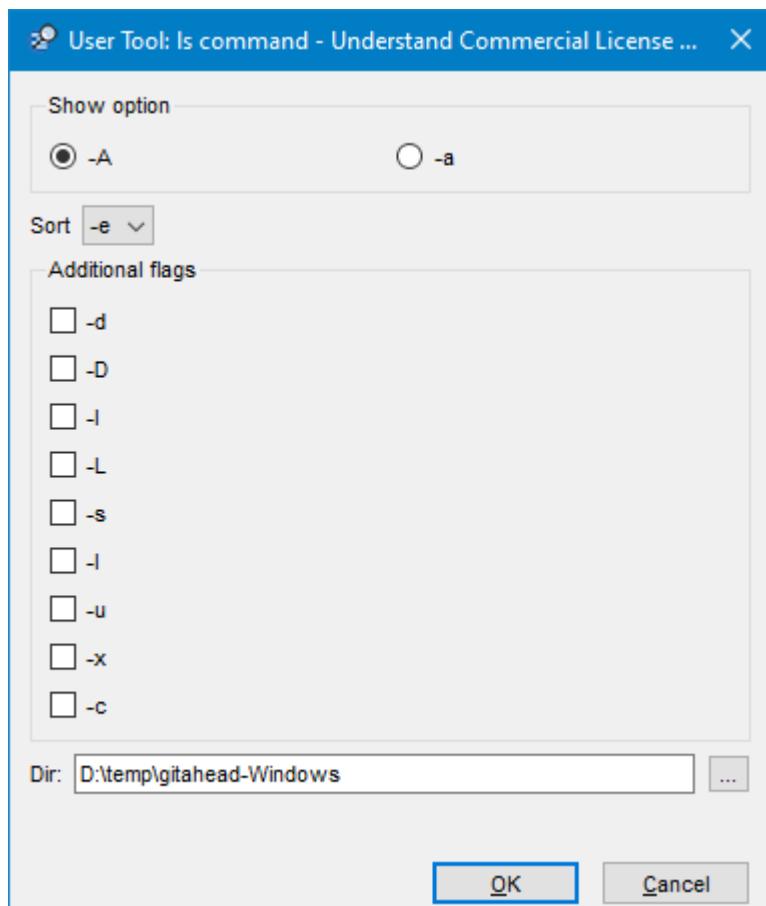
Prompts are processed after the other types of variables, so you can use other variables in the labels and values. For examples, see \$PromptForDir and \$PromptForFile in the previous table.

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:

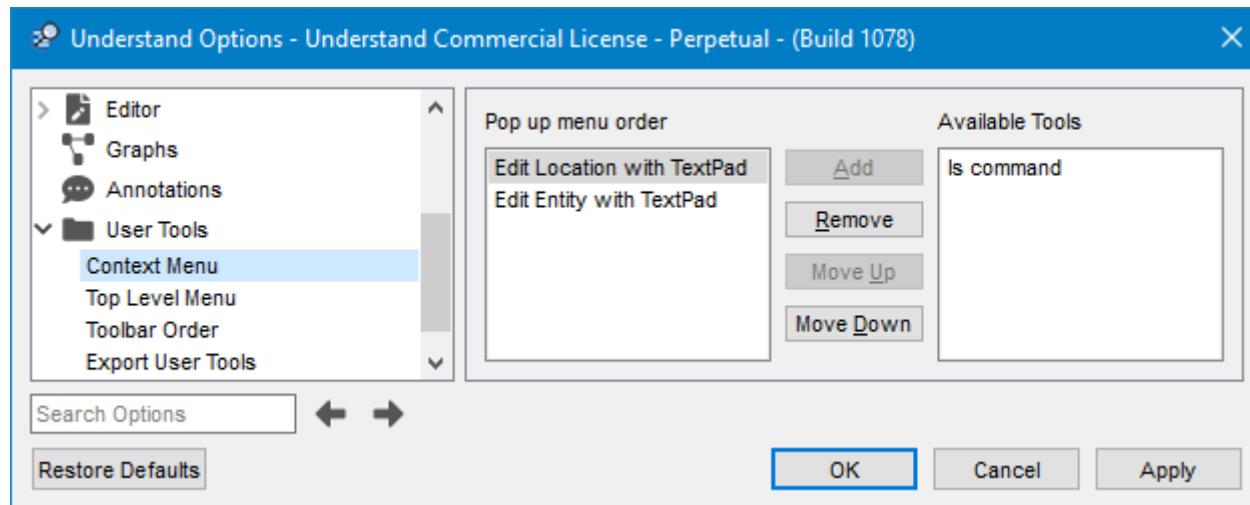
```
$PromptForRadioBoxGH "Show option=-A;-a" $PromptForSelect "Sort=-e;-t" $PromptForCheckBoxGV "Additional flags=-d;-D;-l;-L;-s;-l;-u;-x;-c" $PromptForDir "Dir:=$CurProjectDir"
```



## Adding Tools to the Context Menus

Once a command is defined in the Tools tab, the **Context Menu** category in the Options 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.)

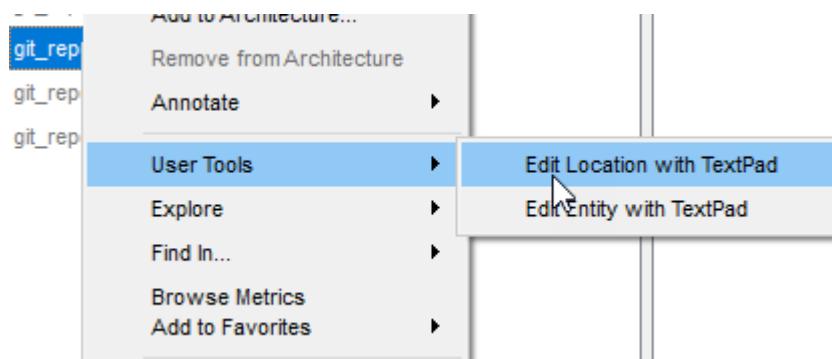
Choose **Tools > User Tools > Configure** and then select the **User Tools > Context Menu** category.



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.

This figure shows a context menu for an entity showing the available external tools.

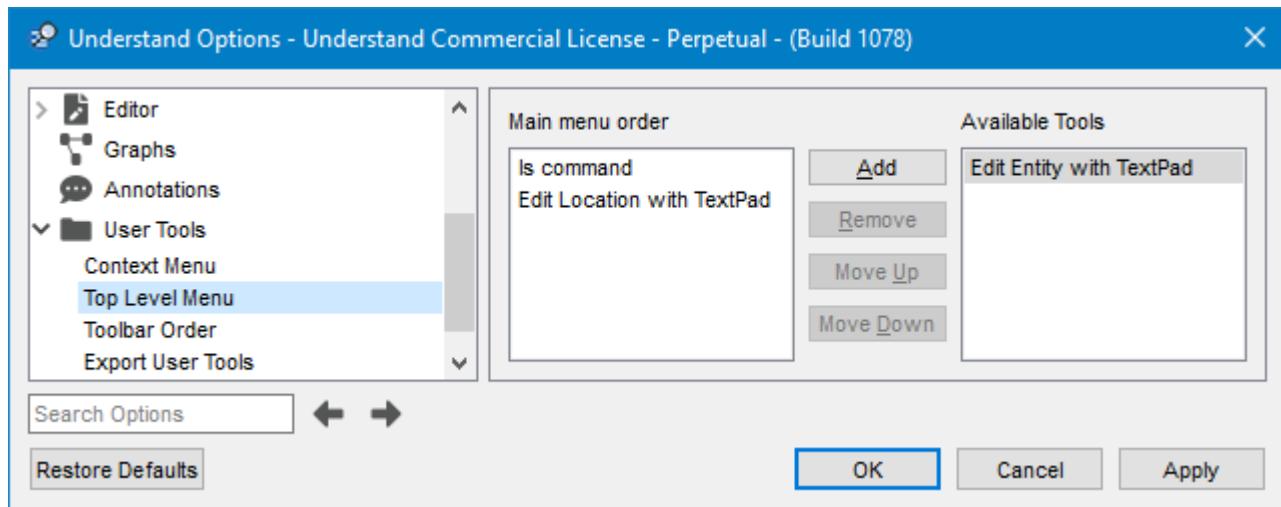


Tools are active or inactive on the context menu based on the context of the parameters provided to the tool. For example, a source editor that specifies \$DeclFile as a parameter is selectable from the context menu for any entity where the declaration is known, but will not be active for an undeclared entity or when no entity is selected.

## Adding Tools to the Tools Menu

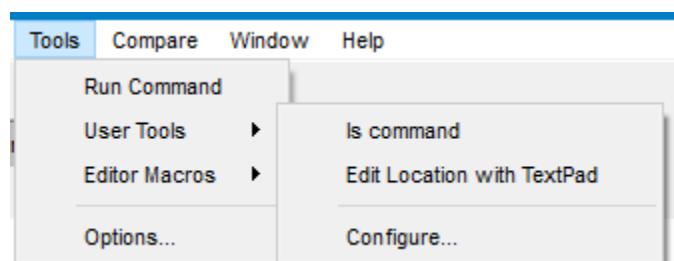
Once a command is defined in the Tools tab, the **Top Level Menu** category in the Options dialog lists user tools that are currently in the **Tools > User Tools** menu on the left and commands you can add to that menu on the right.

Choose **Tools > User Tools > Configure** and then select the **User Tools > Top Level Menu** category.



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**.

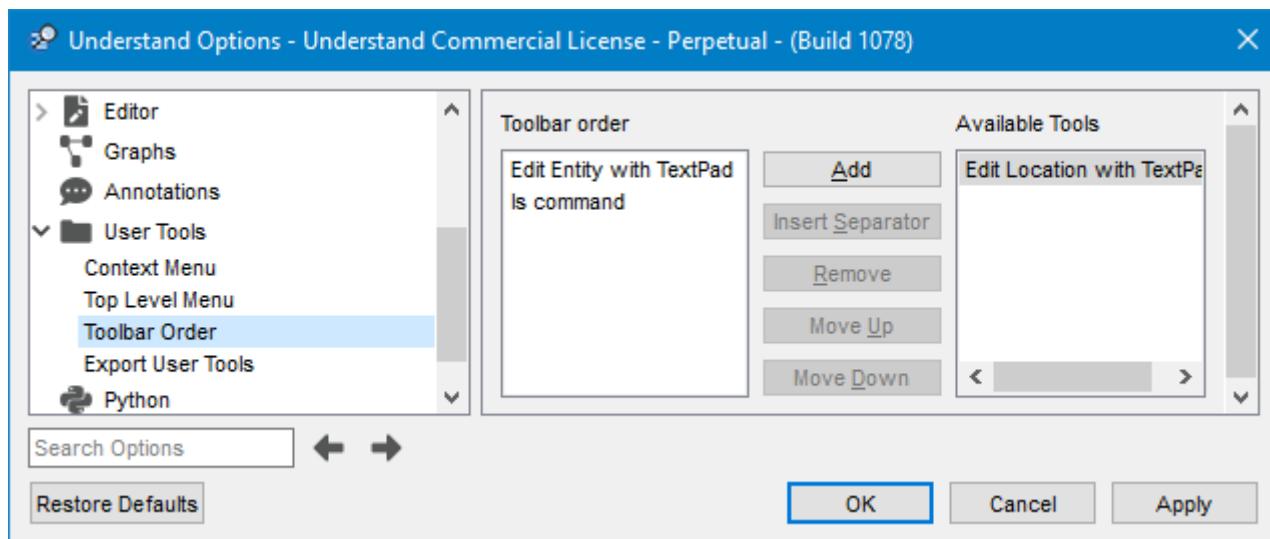
User tools appear on the **Tools** menu in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the tools as desired.



## Adding Tools to the Toolbar

Once a command is defined in the Tools tab, the **Toolbar** category in the Options dialog shows user tools currently in the toolbar in the left box and commands you can add to the toolbar in the right box.

Choose **Tools > User Tools > Configure** and then select the **User Tools > Toolbar Order** category.



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, the following figure shows two user tool icons added between the Graphic View icon and the icons for splitting the workspace. In this case, the first added icon was specified by the user and the second added icon is the default icon for a user tool if no icon is specified.

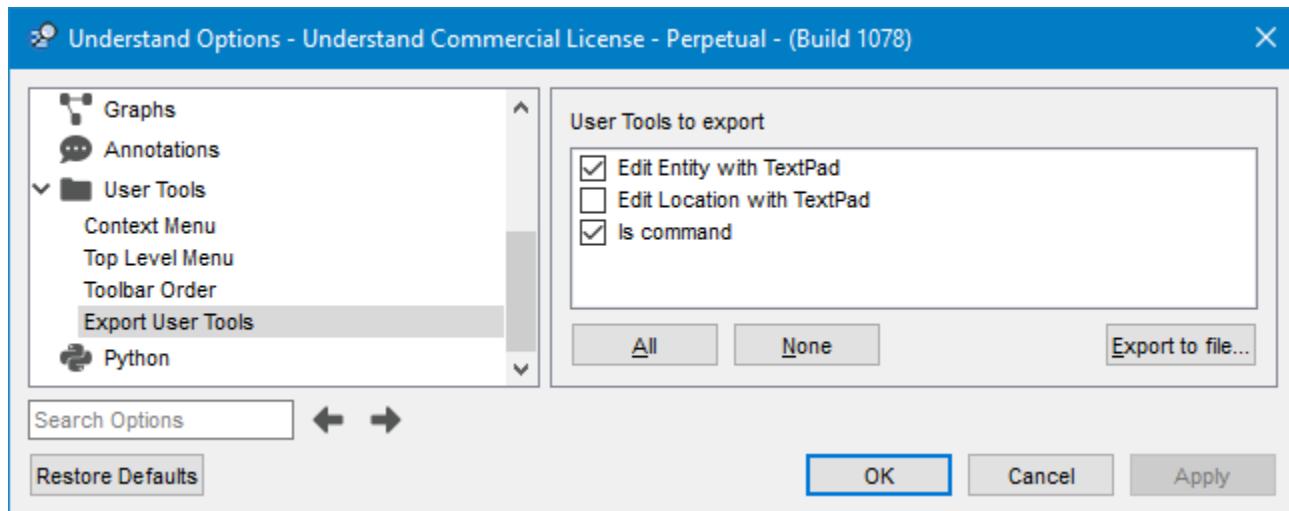


**Note:** You can control which icons are visible in the main toolbar by right-clicking on the background of the toolbar and checking or unchecking items for the various toolbar sections.

## 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** and then select the **User Tools > Export User Tools** category. You will see the following dialog.



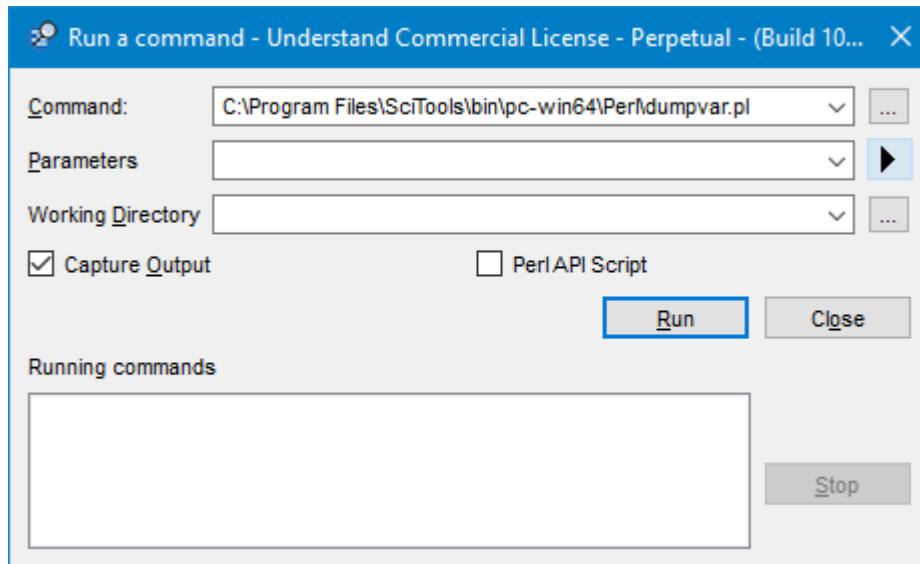
- 2 Check the boxes next to commands you want to share.
- 3 Click **Export to file**.
- 4 Choose a location and filename for an initialization file (\*.ini) that contains the selected user tool information.
- 5 Click **Save**.

To import commands, choose the **User Tools** category in the Options 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 274.
- 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 **Perl API Script** box if this is a script provided by SciTools.
- 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.
- 7 While a command is running, you can select it in the **Running commands** box and click **Stop** to end processing of the command.

The font used in the Command Window is determined by settings in the User Interface category of the Options dialog, which you can open by choosing **Tools > Options** from the menus. See page 94.

On Unix systems, output to both stdout and stderr are captured.

## Running Plugins: Graphs and Interactive Reports

Plugins are scripts that interact directly with *Understand* using one of the *Understand* APIs. Plugins come in two types, graphs and interactive reports. Both can reference the entire project or a specific entity. (See *Writing CodeCheck Scripts* on page 255 for another type of script that uses APIs.)

You can find tutorials and details about plugins and the APIs in the API/Plugins category on the [support website](#).

### APIs

The *Understand* GUI provides a lot of different ways to examine your code, but you may find additional ways you want to access information about your project. For that reason *Understand* has several APIs that let you query the project data. These APIs are read-only; they do not modify the project. If you need to modify the project programmatically, see *Using the und Command Line* on page 287.

You can find tutorials and details about the APIs in the API/Plugins category on the [support website](#). The following APIs are available:

- **Perl API:** This is the most robust API, because it is built into the GUI. If you are writing your own graph and interactive report plugins, this API is recommended. Many sample scripts for the Perl API ship with *Understand* in the scripts folder.
- **Python API:** This API requires a Python 3 installation. It provides full access to the *Understand* database from Python scripts. Several examples ship with *Understand* in the scripts folder.
- **C API:** This API allows you to access data that *Understand* captures from inside your C programs.
- **Java API:** This API is more lightweight; it does not have the full capabilities of the other APIs.
- **.NET API:** This API was developed by one of our users. It is a .NET SDK wrapper that was written in managed C++ (i.e. C++/CLI). Once compiled, it can be accessed by any programming language that targets the .NET framework.

### Interactive Reports

To install a plugin interactive report, either drag the file to the *Understand* window. For example, a plugin that uses the Perl API would be a .upl file. You can also copy plugin files to the following locations:

- **Windows:** C:\Program Files\SciTools\conf\plugin\User\IReport
- **MacOS:** /Users/username/Library/Application Support/SciTools/plugin/IReport
- **Linux:** /home/username/.config/SciTools/plugin/IReport

To run a project-level report, choose **Project > Interactive Reports > plugin\_name** from the menus.

To run an entity-specific report, right-click on the name of the entity almost anywhere in *Understand* and choose **Interactive Reports > plugin\_name** from the context menu.

## Plugin Graphs

To install a plugin graph, either drag the file to the *Understand* window. Plugin graphs should be created using the Perl API and should have a .upl file extension. You can also copy plugin files to the following locations:

- **Windows:** C:\Program Files\SciTools\conf\plugin\User\Graph
- **MacOS:** /Users/*username*/Library/Application Support/SciTools/plugin/Graph
- **Linux:** /home/*username*/.config/SciTools/plugin/Graph

To run a project-level graph, choose **Graphs > Project Graphs > *plugin\_name*** from the menus. Be aware that generating graphs for large projects can be very resource intensive. In some cases the system can be non-responsive for a long period of time while the graphs are generated.

To run an entity-specific graph, select the name of the entity almost anywhere in *Understand* and choose **Graphs > Graphs for *entity\_name* > *plugin\_name*** from the menus. Or, right-click on the entity and choose **Graphical Views > *plugin\_name*** from the context menu.

---

## Chapter 14    Command Line Processing

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* projects 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	287
Using the understand Command Line	295
Using Buildspy to Build Understand Projects	296

## Using the und Command Line

The command-line tool for creating and building *Understand* projects 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 the open project 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 project 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> _
```

For help about interactive mode, use the *und help interactive* command.

- **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 c:/MyProject/MyProject.und
```

The *This.txt* file might contain commands similar to these:

```
# My command file
c:\MyProject\MyProject.und
settings -C++MacrosAdd VERSION="Option_2"
analyze      # update database
report       # generate reports
metrics      # generate metrics
```

For help about batch mode, use the *und help process* command.

- **Line mode:** You can specify a single command or set of commands on a single command line. You must specify the project 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 project, add files, analyze all, and then exit:

```
und create -db c:\myProj\myProj.und -languages c++ add @myFiles.txt analyze -all
und create -languages c++ add @myFiles.txt analyze -all c:\myProj\myProj.und
```

This is the equivalent of running the following set of commands in interactive mode:

```
create -languages c++ c:\myProj\myProj.und  
add @myFiles.txt  
analyze -all
```

Alternately, you could run a sequence of line mode commands like the following:

```
und create -languages c++ c:\myProj\myProj.und  
und add @myFiles.txt c:\myProj\myProj.und  
und analyze -all c:\myProj\myProj.und
```

In general, und commands are case-insensitive.

Und returns a value of 1 if an error occurred.

Und supports the following global options that can be added to any command:

Option	Discussion
-db	Specify the project to use
-ini <i>filename</i>	Specify a configuration file 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 289
analyze	Analyzes the project files	page 293
codecheck	Runs CodeCheck	page 294
convert	Convert a legacy .udb project to a .und project	page 289
create	Creates an empty project	page 289
export	Exports settings, dependencies, or architectures	page 292
help	Gives help information for a command	page 289
import	Imports project settings and architectures	page 292
license	Provide information about the software license	page 291
list	Lists information about the project	page 291
metrics	Generates project metrics	page 293
process	Runs all the commands in a text file in batch mode	page 287
projectinfo	Provide information about the project	page 291
purge	Purges the project database	page 293
remove	Removes files, directories, roots, and architectures	page 291
report	Generates project reports	page 293

Option	Discussion	See
settings	Sets project settings and overrides	page 292
uperl	Runs Perl scripts	page 294
version	Shows the current software version	page 289

Refer to the sections that follow for details on the commands supported by und. Additional information about the und command line is provided in the “Using Understand from the Command Line with Und” topic on the [Support website](#).

## Getting Help on Und

Since we do frequent builds of *Understand*, this manual does 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 project. Specify the name of the project either with the -db option or as the last parameter. Any settings allowed with the settings command (see page 292) can also be used with create. For example:

```
und create -db c:\newProj\newProj.und -languages c++ c#
und create -open_files_as_read_only on c:\newProj\newProj.und
```

For more information, use the following command:

```
und help create
```

## Converting a Legacy Project

To convert a legacy .udb project to the new .und directory project storage format, use the und convert command. For example:

```
und convert legacyDatabaseName.udb newDatabaseName.und
```

If no new *projectName.und* location is provided, the existing .udb path and name is used. The -override option overwrites the new .und project if exists. The -deleteudb option deletes the old .udb project file if the conversion was successful.

For more information, use the following command:

```
und help convert
```

## 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 wildcard abilities of your operating system shell. For example:

```
und -db c:\myProj\myProj.und add \usr\myproject
und -db c:\myProj\myProj.und add file1.cpp file2.cpp
und -db c:\myProj\myProj.und 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. If wildcards (for example, proj\*.c) do not match the correct list of files or 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 c:\myProj\myProj.und add @myfiles.list
```

Note that there is no limit on the number of files listed in the list file.

Another way to add files to a project is to add the files and file override settings already configured in one project to another project. The command format for this is:

```
und add c:\srcProj\srcProj.und c:\destProj\destProj.und
```

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.

**Exclude strings** are processed relative to the top-level directory passed to the *add* command, and are applied to all files in all subdirectories. The exclude strings are internally processed as follows:

- 1 Separate the exclude string into the list of wild cards based on spaces, commas, and semicolons.
- 2 For each separate exclude string, replace forward and back slashes with the pattern [\\], which matches either slash.
- 3 Prepend the absolute path of the top-level directory to the wild card, ensuring that [\\] separates the path from the initial wild card.
- 4 Compare the wild card to both the file short name and file long name for every file below that top-level directory. The comparison is done with QRegExp wild card matching.

Named roots definitions on the “und” command line have the highest precedence. The next precedence is named roots defined as environment variables at the operating system level, and finally by named roots defined in the *Understand* project configuration. See page 52.

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 c:\myProj\myProj.und
und remove C:\SomeDirectory c:\myProj\myProj.und
und -db c:\myProj\myProj.und remove vs1.vcproj vs2.vcproj
und remove -file main.c c:\myProj\myProj.und
```

For more information, use the following command:

```
und help remove
```

## Getting Information about a Project and the License

Use the `und projectinfo` command to show the version of *Understand* last used to modify the project and when, the full path to the project directory, the last analysis date and time, the versions of *Understand* used to create and modify the project, and the directory where user-specific information is stored.

Use the `und license` command to get the license code, return code, and expiration date for the installation of *Understand*. See the “Command Line Licensing” topic on the [Support website](#) for more about command-line access to license information.

Use the `und list` command to list file, setting, architecture, or named root settings in a project. For example:

```
und list -tree files c:\myProj\myProj.und
und list settings c:\myProj\myProj.und
und list arches c:\myProj\myProj.und
und list roots c:\myProj\myProj.und
```

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 c:\myProj\myProj.und
und list -override @listfile.txt c:\myProj\myProj.und
und list -metrics -reports settings c:\myProj\myProj.und
und list -all settings c:\myProj\myProj.und
und list -lang C++ -macros -includes settings c:\myProj\myProj.und
und list -lang fortran settings c:\myProj\myProj.und
```

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 c:\myProj\myProj.und
```

For example, the following command adds the specified directory to the list of C/C++ include directories in the project:

```
und settings -c++includesadd c:\myincludes c:\myProj\myProj.und
```

In general, setting names are the same as the field name in *Understand*, but with spaces omitted. For example:

```
und settings -ReportDisplayCreationDate on c:\myProj\myProj.und  
und settings -ReportFileNameDisplayMode full c:\myProj\myProj.und  
und settings -ReportReports "Data Dictionary" "File Contents" c:\myProj\myProj.und  
und settings -C++MacrosAdd MYLONG="Long Text" c:\myProj\myProj.und  
und settings -ReportNumberOfPages 250 c:\myProj\myProj.und
```

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 project to import setting that you have exported from another project.

For example:

```
und import settings.xml c:\myProj\myProj.und  
und import -arch myArch.xml c:\myProj\myProj.und
```

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 c:\myProj\myProj.und
```

This command exports architectures to an XML file that you can use with the `und import` command:

```
und export -arch "Calendar" toHere.xml c:\myProj\myProj.und
```

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 c:\myProj\myProj.und  
und export -dependencies class matrix output.csv c:\myProj\myProj.und  
und export -dependencies arch myArch csv output.csv c:\myProj\myProj.und  
und export -dependencies -col refs -format short file csv out.csv c:\myProj\myProj.und
```

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 c:\myProj\myProj.und
und analyze -files @someFile.txt
und -db c:\myProj\myProj.und analyze -rescan -changed
und analyze -files file1.cpp file2.cpp c:\myProj\myProj.und
und -db c:\myProj\myProj.und -rescanwithoutanalyze
```

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 analyze 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 analyzed data from the Understand project, leaving only the project definition. This significantly shrinks the project file size, which you may want to do before sharing the project or backing it up. Running the `analyze` command will repopulate the project. For example:

```
und purge c:\myProj\myProj.und
```

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 291), and changed using the `settings` command (see page 292). For example:

```
und list -reports settings c:\myProj\myProj.und
und report c:\myProj\myProj.und
```

**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 c:\myProj\myProj.und
und metrics -arch myArch c:\myProj\myProj.und
und metrics -html arch1 arch2 c:\temp c:\myProj\myProj.und
```

This command uses the current metrics settings, which can be viewed by using the `und list` command (see page 291), and changed using the `settings` command (see page 292). For example:

```
und list -metrics settings c:\myProj\myProj.und
```

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 c:\myProj\myProj.und
```

You can create a CodeCheck configuration file as described on page 243.

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. You can also export the list of checks performed and the ignored violations to a file without running the CodeCheck configuration. For example, the following command runs the specified configuration file on the files listed in `filelist.txt` and generates both the HTML and CSV versions of the results:

```
und codecheck -html -files filelist.txt config.ini C:\temp c:\myProj\myProj.und
```

For more information, see the “Running Codecheck from the Command Line” topic on the [Support website](#) and 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 c:\myProj\myProj.und
```

For more information, use the following command:

```
und help uperl
```

Note that the `und uperl` command does not support any graphical uperl commands, such as `$ent->draw`.

## 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 c:\myProj\myProj.und
```

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.
-help	Open a command line help window for the understand command.
-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.)
-quiet_startup	Use this option to disable all dialogs and splash screens shown during startup.
-resetlicenseconfig	Clean out old licenses from the licensing file.
-SlowConnect	Use a longer timeout when waiting to connect to the license server. Use this option if you have a slow connection.
-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. (This step is optional; Buildspy can be run incrementally to update only the files it is run on.)
- 4 From the directory where your make file is located, run a command similar to the following:

```
buildspy -db path/name.und -cmd <compile_command>
```

For example:

```
buildspy -db c:\myProj\myProj.und -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.

The wrappers work with any compiler that has gcc-like syntax. You can use any of the following methods to specify which compilers `gccwrapper` and `g++wrapper` should call:

- Use Buildspy's `-cc` and/or `-cxx` command line arguments.
- Define the `UND_PBCCCOMPILER` and `UND_PBCXXCOMPILER` environment variables. These environment variables are checked whenever `gccwrapper` and `g++wrapper` are run.
- Edit the configuration file located in `$HOME/.config/SciTools` on Linux systems and `$HOME/Library/Preferences` on Mac.

For more information about using Buildspy, use the `buildspy -help` command and see the Buildspy topic on the [Support website](#).

---

## Chapter 15    Quick Reference

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	298
Edit Menu	299
Search Menu	299
View Menu	300
Project Menu	300
Architectures Menu	301
Metrics Menu	301
Graphs Menu	301
Checks Menu	302
Annotations Menu	302
Tools Menu	302
Compare Menu	303
Window Menu	303
Help Menu	304

---

## File Menu

The **File** menu in *Understand* contains the following commands:

Command	See
New > Project	page 34
New > File	page 183
Open > Project	page 19
Open > Legacy .udb Project	page 19
Open > File	page 183
Close <i>project_name</i>	page 19
Export to Image File	page 236
Save <i>filename</i>	page 171
Save <i>filename</i> As	page 171
Save All	page 171
Page Setup	page 191
Print <i>filename</i>	page 191
Print Entity Graph	page 238
Recent Files	page 97
Recent Files > Clear Menu	page 97
Recent Projects	page 97
Recent Projects > Clear Menu	page 97
Exit	page 19

## Edit Menu

The **Edit** menu in *Understand* contains the following commands:

Command	See
Undo	page 170
Redo	page 170
Cut	page 169
Copy	page 169
Copy Image to Clipboard	page 236
Paste	page 169
Select All	page 169
Comment Selection	page 182
Uncomment Selection	page 182
Change Case	page 181
Toggle Overtype	page 182
Zoom > Zoom In	page 166
Zoom > Zoom Out	page 166
Zoom > Reset Zoom	page 166
Fold All	page 180
Soft Wrap	page 182
Hide Inactive Lines	page 180

## Search Menu

The **Search** menu in *Understand* contains the following commands:

Command	See
Find	page 163
Find Previous	page 163
Find & Replace	page 163
Go to Line	page 168
Go to Matching Brace	page 180
Instant Search	page 148
Find in Files	page 150
Replace in Files	page 153
Show Search Results	page 152
Find Entity	page 155

## View Menu

The **View** menu in *Understand* contains the following commands:

Command	See
Toolbars	page 162
Browse Mode	page 169
Analysis Log	page 88
Bookmarks	page 184
Contextual Information	page 164
Dependency Browser	page 135
Entity Filter	page 124
Entity Locator	page 155
Favorites	page 143
Information Browser	page 126
Metrics Browser	page 208
Previewer	page 179
Project Browser	page 132
Scope List	page 167
Window Selector	page 159

## Project Menu

The **Project** menu in *Understand* contains the following commands:

Command	See
Overview	page 207
Configure Project	page 39
Analyze Changed Files	page 88
Analyze All Files	page 88
Improve Project Accuracy > Undefined Macros	page 90
Improve Project Accuracy > Missing Includes	page 91
Improve Project Accuracy > More Information	page 90
Export Dependencies > Architecture Dependencies	page 139
Export Dependencies > File Dependencies	page 139
Export Dependencies > Class Dependencies	page 139
Interactive Reports	page 284

---

## Architectures Menu

The **Architectures** menu in *Understand* contains the following commands:

Command	See
Browse Architectures	page 194
Design Architecture	page 199
Manage Orphans	page 190
Add <entity> to Architecture	page 198
Remove <entity> from Architecture	page 197

---

## Metrics Menu

The **Metrics** menu in *Understand* contains the following commands:

Command	See
Browse Metrics	page 208
Export Metrics	page 210
Metrics Treemap	page 213

---

## Graphs Menu

The **Graphs** menu in *Understand* contains the following commands:

Command	See
Dependency Graphs > By architecture	page 201
Dependency Graphs > Load Saved Dependency Graph	page 222
UML Class Diagram > By architecture	page 220
Project Graphs	page 284
Graphs for selected entity	page 222

## Checks Menu

The **Checks** menu in *Understand* contains the following commands:

Command	See
Open CodeCheck	page 242
Checks Selection	page 243
Re-Run <checks>	page 242
Analyze Changes and Re-Run <checks>	page 242
Configure Dependency Checks	page 203
Run Dependency Checks	page 203

## Annotations Menu

The **Annotations** menu in *Understand* contains the following commands:

Command	See
Annotation Viewer	page 186
Browse All Annotations	page 187
View <file> Annotations	page 186
Configure Annotations	page 118
Manage Orphans	page 190
Annotate	page 188

## Tools Menu

The **Tools** menu in *Understand* contains the following commands:

Command	See
Run Command	page 283
User Tools > <i>tool_name</i>	page 280
User Tools > Configure	page 272
Editor Macros > Record Macro	page 183
Editor Macros > Replay Macro	page 183
Editor Macros > Save Macro	page 183
Editor Macros > Configure Macros	page 110
Options	page 93

---

## Compare Menu

The **Compare** menu in *Understand* contains the following commands:

Command	See
Compare Projects	page 263
Locate Changed Entities	page 263
Compare Files/Folders	page 259
Compare Entities	page 262
Compare Arbitrary Text	page 263

---

---

## Window Menu

The **Window** menu in *Understand* contains the following commands:

Command	See
Close <i>current_file</i>	page 171
Close All Document Windows	page 159
Session Browser	page 162
Release Window	page 159
Split Workspace Vertically	page 159
Split Workspace Horizontally	page 159
Unsplit Workspace	page 159
Reset Windows to Default Layout	page 162
< <i>open source file list</i> >	page 159
Windows Navigator	page 159

---

## Help Menu

The **Help** menu in *Understand* contains the following commands:

Command	See
Help Content	page 14
Key Bindings	page 101
Example Projects	page 14
Perl API Documentation	page 30
Python API Documentation	page 30
Frequently Asked Questions	page 14
Show Welcome Page	page 19
Licensing	page 14
Privacy	page 14
Reset All Hints	page 14
About Understand	page 14

# Index

## Symbols

: in F77 identifiers, Fortran 75  
? in regular expressions 158  
? wild card 158  
. in regular expressions 158  
" prefixing octal constants or string literals, Fortran 75  
"" surrounding normal includes 69  
[ - ] in regular expressions 159  
[ ] in regular expressions 159  
[^ ] in regular expressions 159  
\* comments field 75  
\* in regular expressions 158  
\* wild card 158  
/\* ... \*/ C-style comments 66, 75  
\ in regular expressions 159  
\< in regular expressions 158  
\> in regular expressions 158  
# comment in command line file 294  
^ in regular expressions 158  
+ expanding tree in Information Browser 127  
+ in regular expressions 158  
<> surrounding system includes 69  
= macro definition in command line 294  
| in regular expressions 159  
\$ in regular expressions 158  
\$ prefixing external tool parameters 273, 274

## A

About Understand option, Help menu 14  
Activate when Control key is pressed field, Browse Mode Editor options 112  
actual parameters, relationship to formal parameters, Ada 55  
Ada configuration for 54  
macro definitions 56  
versions supported 13  
Ada category, Project Configuration dialog 54  
Add found include files to source list field, C++ Includes 69  
Add found system include files to source list field, C++ Includes 69  
-addDir option, und command 289

-addFiles option, und command 290  
Adobe Acrobat, saving graphical views to 239  
Advanced category, Editor options 107  
Alerts category, User Interface options 98  
Allow 75  
Allow Colons in Names field, Fortran 75  
Allow C-style comments field, Fortran 75  
Allow embedded SQL field, Pascal 81  
Allow function declaration without parentheses field, Fortran 75  
Allow interactivity during intensive processing field, General options 95  
Allow Nested Comments field, C++ 66  
Allow parameter declaration without parentheses field, Fortran 75  
Allow quote in octal constants field, Fortran 75  
Analysis Log option, View menu 89  
Analysis Log window 89  
    reopening previous analysis Log 161  
    saving to text file 89  
Analysis Options area 273  
Analyze All Files option, Project menu 88  
Analyze category, Understand Options dialog 103  
Analyze Changed Files option, Project menu 88  
-analyze option, und command 293  
Analyze unincluded headers in isolation field 69  
-analyzeFiles option, und command 293  
analyzing projects 88  
    after changing configuration of 40  
    beep on completion of 103  
    on command line 293  
and operators, including in strict complexity, Ada 55  
angle brackets (<>) surrounding system includes 69  
Animate windows/drawers field, User Interface options 96  
Annotations category, Project Configuration dialog 118  
Annotations menu 302  
Apache Lucene syntax 148  
APIs 30  
Append the names of externally linkable entities with field  
    C++ 66  
    Fortran 76  
Application font field, General options 94  
Architecture Browser 28, 194  
    duplicating architectures 196  
    graphical views in 195, 201

---

metrics in 196, 204, 212  
opening Architecture Builder from 195  
renaming architectures 196  
right-clicking in 196  
Architecture Builder 195  
architectures 17, 193  
auto-architectures 195  
duplicating 196  
editing. *See* Architecture Builder  
exporting dependency list 139  
graphical views of 195, 201  
hierarchies of, exploring 194  
list of 194  
listing 28  
metrics about 196, 204, 212  
navigating 28  
renaming 196, 197  
Architectures menu 301  
arrows, for Information Browser history 131  
ASP style tags 87  
assembly  
    configuration for 58  
    embedded in C code 67, 72  
    include files, directories for 58  
Assembly category, Project Configuration dialog 58  
asterisk (\*)  
    in regular expressions 158  
    wild card 158  
Auto Category, C++ Includes 70  
auto-architectures 28, 195  
Auto-complete fields, Advanced Editor options 109  
Auto-indent fields, Advanced Editor options 109, 110  
Automatic compool file field, JOVIAL 80

## B

background processing, interactivity during 95  
backslash (\) in regular expressions 159  
Basic  
    configuration for 86  
beep on analysis completion 103  
bitmaps, saving graphical views as 236  
blocks  
    expanding and collapsing 166, 180  
    in Filter Area 23  
blue background for text 111  
blue text 111

bookmarks  
    creating 184  
    list of 161, 184  
    navigating 184  
Bookmarks option, View menu 161, 184  
Boolean searches 149  
braces, matching. *See* brackets  
brackets  
    in regular expressions 159  
    matching 180  
Browse Architectures option, Project menu 194  
Browse Metric Charts option, Metrics menu 214  
Browse Metrics option, Metrics menu 206, 208  
Browse Mode option, View menu 169  
Browse Mode, Source Editor 169  
Browser Area 17  
Build Log 33  
Buildspy 296

## C

C/C++  
    API for custom reports 30, 206  
    configuration for 63, 66, 73  
    include files, auto including 70  
    include files, directories for 68  
    include files, ignoring 70  
    macro definitions 71  
    preprocessor directives 71  
    strict analysis 63, 73  
    versions supported 13

C#  
    configuration for 73  
    reference files, importing 73  
    versions supported 13

C# category, Project Configuration dialog 73  
C++ (Strict) category, Project Configuration dialog 63, 73  
C++ category, Project Configuration dialog 66  
caching include files, C++ 68  
Calendar auto-architecture 28, 195  
Called by menu, graphical views 229  
capitalization, of selected text 181  
caret (^) in regular expressions 158  
Caret Line field, Editor options 106  
case  
    changing for selected text 181  
    of displayed entity names, Ada 56

of displayed entity names, Fortran 76  
 of displayed entity names, JOVIAL 80  
 of displayed entity names, Pascal 82  
 of displayed entity names, PL/M 83  
 of externally linkable entities, Ada 55, 80  
 of externally linkable entities, Fortran 75  
 of externally linkable entities, Pascal 81  
 of identifier names, Fortran 75  
**Case of externally linkable entities field**  
 Ada 55, 80  
 Fortran 75  
 Pascal 81  
**Case sensitive identifiers field, Fortran 75**  
**case sensitivity**  
 of comparison 270  
 of Fortran identifiers 75  
**Change Case menu option 181**  
**Change Case option, Edit menu 181**  
**Change Log 33**  
 character strings, blue text for 111  
**Checks menu 302**  
**CIS. See Contextual Information Sidebar**  
 .class files 78  
**Class Paths category, Java 78**  
**classes**  
 exporting dependency list 139  
 extended by other classes 230  
 extending other classes 230  
 implemented 231  
 implemented by 231  
 listing in Filter Area 23  
 providing without source code, Java 78  
**clipboard**  
 copying graphical view to 236  
 copying text from Source Editor to 169  
**Close All Document Windows option, Window menu 159, 171**  
**Close option**  
 File menu 20  
 Window menu 159, 171  
**Close Selected Window(s) command 161**  
**Cmake import 49**  
**COBOL**  
 configuration for 60  
 copybook files, directories for 61  
**COBOL category, Project Configuration dialog 60**  
**CodeCheck 240**  
 checks 243  
 configuration 243  
 result log 250  
 running 242  
 scripts 255  
**colon (:) in F77 identifiers, Fortran 75**  
**Color Mode field, Advanced Editor options 107**  
**colors**  
 in comparison 270  
 for printing source code 107  
 of rows in User Interface, alternating 100  
 in Source Editor, customizing 166, 111  
 in Source Editor, default 111  
**column truncation**  
 Fortran 75  
 JOVIAL 79  
**command line. See und command; understand command**  
**Command window 283**  
**Command Window Font 97**  
**commands, external. See external tools**  
**Comment Selection menu option 182**  
**comments**  
 adding or removing 182  
 associating with entities, Ada 56  
 associating with entities, C 65  
 associating with entities, C++ 67  
 associating with entities, Java 77  
 C-style 66  
 green text for 111  
 nested, C++ 66  
**companion file 183**  
**Compaq Pascal 81**  
**Compare Arbitrary Text option 263**  
**Compare Arbitrary Text option, Tools menu 263**  
**Compare Entities option, Compare menu 262**  
**Compare files by content field 69**  
**Compare Files/Folders option, Compare menu 259**  
**Compare menu 303**  
 Compare Entities option 262  
 Compare Files/Folders option 259  
**compilation environment, Ada 55**  
**compilation unit**  
 With By relationships of 235  
**compiler**  
 COBOL 60  
 compared to Understand 32

---

Compiler field  
    C++ 66  
    PL/M 83

Compiler Include Paths field, C++ 66

complexity  
    exception handlers included in, Ada 55  
    FOR-loops included in, Ada 55  
    strict, and/or operators in, Ada 55

Component field, Key Bindings options 101

compool file, JOVIAL 80

Configure option, Project menu 39

constants  
    displayed in graphical views 230

Constants menu, graphical views 230

contact information 13

Context Browser, Contextual Information Sidebar 164

-contextmenu option, understand command 295

Contextual Information option, View menu 164

Contextual Information Sidebar (CIS) 164

!COPY directives, directories to search, JOVIAL 80

Copy category, JOVIAL 80

Copy option, Edit menu 169

Copybook category 61

copybook files  
    adding to project, COBOL 61

copying text  
    rectangular area 169

Count and/or operators in strict complexity field, Ada 55

Count exception handlers in complexity field, Ada 55

Count for-loops in complexity field, Ada 55

Create and cross-reference record object components  
    field, Ada 55

Create implicit special member functions field 67

Create references in inactive code field, C++ 67

Create references in inline assembly, C++ 67

Create references to local objects field, C++ 67

Create references to macros during macro expansion  
    field, C++ 67

Create references to parameters field, C++ 67

Create relations between formal and actual parameters  
    field, Ada 55

Crossing layout option 232

cross-referencing record object components, Ada 55

CSS files 13

C-style comments  
    in Fortran 75  
    nesting of, allowing 66

CSV file  
    exporting dependencies to 139  
    exporting metrics to 206, 210

Ctrl+Alt+F keystroke, find and replace text 163

Ctrl+F keystroke, find text 163

Ctrl+j keystroke, jump to matching bracket 180

Ctrl+right-click keystroke  
    creating new windows 21, 122

Ctrl+Shift+j keystroke, select text in brackets 180

\$Cur variables 273, 274

Cut option, Edit menu 169

-cwd option  
    understand command 295

Cytoscape  
    installation location 104  
    XML output 139

Cytoscape XML 139

## D

dark mode 95

database 17  
    changes to format of 33  
    file extension for 17, 33  
    multi-user read/write access for 33  
    *See also* project

Date Format field 108

-db option  
    understand command 295

DEC Pascal 81

\$Decl variables 273, 274, 275

declaration views  
    calling methods displayed in 229  
    constants displayed in 230  
    default members displayed in 230  
    extended by classes displayed in 230  
    extended classes displayed in 230  
    external functions displayed in 230  
    file dependencies displayed in 230  
    globals displayed in 231  
    header files include by's displayed 231  
    implemented by classes displayed in 231  
    implemented classes displayed in 231  
    imported entities displayed in 231  
    include files displayed in 231  
    inherited entities displayed in 231  
    invocations displayed in 232  
    local items displayed in 232

- members displayed in 232
  - objects displayed in 232
  - operators displayed in 232
  - private members displayed in 233
  - protected members displayed in 233
  - public members displayed in 233
  - rename declarations in 233
  - static functions displayed in 234
  - types displayed in 234, 235
  - used-by items shown in 235
  - variables displayed in 235
  - With By relationships displayed in 235
  - With relationships displayed in 235
    - See also* graphical views
  - declarations
    - local, including in project, C++ 67
  - Default Members menu, graphical views 230
  - Default style field, Editor options 105
  - default working directory 95
  - Delphi Pascal 81
  - dependencies
    - browsing 135
    - exporting 136, 139
    - file 230
  - Dependency Browser 135
  - Dependency category, Understand Options dialog 104
  - Dependency Checks
    - configuring 203
    - exporting 204
    - running 204
  - Dependency Graph 195
  - Dependency Graphs option, Graphs menu 201
  - Dependent menu
    - graphical views 230
  - Dependent Of menu, graphical views 230
  - DFM converter exe field 81
  - diff option, understand command 295
  - Different Word Color in comparison 270
  - directives, C++ 71
  - directories
    - adding to project 43
    - comparing 259
    - copying 261
    - deleting from project 44
    - overriding settings for 45
    - watched, setting 44, 45, 46
  - Directory Structure auto-architecture 28, 195
  - Display entity names as field
    - Ada 56
    - Fortran 76
    - JOVIAL 80
    - Pascal 82
    - PL/M 83
  - docking windows 16
  - Document Area 17
  - documentation 14
  - dollar sign (\$)
    - in regular expressions 158
    - prefixing external tool parameters 273, 274
  - DOS line termination style
    - for saving source files 106
  - DOT files, saving graphical views as 237
  - double quotes. *See* quotes
  - double-clicking
    - in Bookmarks area 184
    - column header dividers 155
    - entities in Entity Filter 129
    - entities in Information Browser 129
    - in Exploring View 134
    - messages in Analysis Log window 89
    - in Project Browser 208
    - in Search Results window 152
    - in Source Editor 27
  - drawers 16, 96
  - Drill down command 215
  - drop-down menu *See* right-clicking
  - Duplicate Architecture menu option 196
  - duplicate references, C++ 68
  - Duplicate Subtrees menu, graphical views 230
- E**
- edges, in dependency graphs 227
  - Edit Architecture menu option 195
  - Edit Graphic Filters menu option 225
  - Edit menu 299
    - Change Case option 181
    - Copy option 169
    - Cut option 169
    - Select All option 169
    - Toggle Overtype option 182
  - Editor category, Understand Options dialog 105
  - Editor Macros option, Tools menu 183
  - editor windows, saving automatically 94
  - editor, external 113

---

editor, source. *See Source Editor*  
Emacs editor 113  
embedded SQL, in Pascal 81  
Enable Editor Tooltips field 112  
Enable permissions checking for NTFS filesystems field 95  
encoding formats 48, 106  
entities 17  
    comments associated with, Ada 56  
    comments associated with, C 65  
    comments associated with, Java 77  
    comparing two entities 262  
    current, information about 164  
    displaying source for. *See Source Editor*  
    favorites of, displaying 143  
    favorites of, marking 142  
    full name of, displaying 125  
    graphical views of. *See graphical views*  
    hierarchy of, exploring 134, 164  
    information about. *See Information Browser*  
    listed in Entity Filter 124  
    listed in Entity Locator 24, 155  
    listed in Filter Area 23  
    metrics for. *See metrics*  
    references for 130  
    relationships between 17  
    sorting of 124  
    unknown 235  
    unresolved 235  
    *See also specific entities*

Entity Filter 124  
    displaying information about entities in 126  
    displaying source of selected entity 129  
    entities not listed in 24, 155  
    jumping to entities in 23  
    location of 17  
    right-clicking in 22

Entity Locator 24, 155  
    column headers in, customizing 156  
    columns in, hiding and reordering 156  
    columns in, resizing 155  
    filtering by selection 157  
    filtering manually 158  
    filtering with regular expressions 158  
    filtering with wildcards 158  
    length of names in 155  
    opening 155

    right-clicking in 24, 155  
    right-clicking on column header 156, 158  
    right-clicking on entities 157  
    sorting entities in 156

Entity Locator option, View menu 24, 155  
environment variables  
    using in include paths, assembly 59, 61  
    using in include paths, C++ 69

equal sign (=) in macro definition in command line 294

errors  
    displaying from Analysis Log window 89  
    parse errors, prompting for, Ada 56  
    parse errors, prompting for, Fortran 76

Example Projects option, Help menu 14

exceptions  
    handlers for, including in Ada complexity 55

-existing option, understand command 295

Exploring view 134

Export Dependency CSV 139

Export Metrics option, Metrics menu 206, 210

exporting tool commands 282

Extended By menu, graphical views 230

Extends menu, graphical views 230

extensions. *See file extensions*

external editor 113

External Editor category, Editor options 113

External Functions menu, graphical views 230

external tools  
    adding to Right-click Menu 274, 279  
    adding to toolbar 281  
    adding to User Tools menu 280  
    commands for, importing and exporting 282  
    commands for, running 283  
    configuring 272  
    editor 113

externally linkable entities  
    case of, Ada 55, 80  
    case of, Fortran 75  
    case of, Pascal 81  
    prefix for 66, 76, 82  
    suffix for, C++ 66  
    suffix for, Fortran 76

Extract Function refactoring 175

Extract Temp refactoring 178

**F**

F5 key, Find in Files option 150

- favorites 25  
     displaying 143  
     marking entities as 142  
 Favorites option, View menu 25, 143  
 file extensions  
     configuring for project 47  
     for database 17, 33  
 File Information, Contextual Information Sidebar 164  
 File menu 298  
     Close option 20  
     New > File option 183  
     New > Project option 34  
     Open > File option 183  
     Open > Legacy .udb Project option 19  
     Open > Project option 19  
     Page Setup option 191  
     Print Entity Graph option 238  
     Print option 191  
     Recent Projects option 19  
     Save All option 171  
     Save option 171  
 File Mode field, Editor options 106  
 File Options category, Project Configuration dialog 48  
 file permission checking 95  
 File Sync box 129  
 File Types category, Project Configuration dialog 47  
 Filename menu, graphical views 230  
 filenames  
     in graphical views 230  
     in title areas 97  
 files  
     comparing 259  
     copying 261  
     exporting dependency list 139  
     Information Browser for 164  
     location included in listing of 125  
     searching 25, 150  
     searching and replacing in 153  
     toolbar for 162  
 Files category, Project Configuration dialog 42  
 Filter Area 23  
     See also Entity Filter; Project Browser  
 Filter By Selection menu option 157  
 Filter field, Entity Filter 124  
 filters  
     for graphical views 225  
     See also Entity Filter; Entity Locator
- Find dialog 163  
 Find Entity option, Search menu 155  
 Find in Files dialog 17, 25  
 Find in Files menu option 25, 150  
 Find option, Search menu 163  
 Find Results window 152  
 fixed file format, Fortran 75  
 folders. See directories  
 folding code 180  
 font  
     Command Window 97  
     for printing source code 107  
     for Source Editor windows 105  
     in Source Editor windows 180  
     in Understand, changing 94  
 Font Size field, Advanced Editor options 107  
 FOR-loops, including in complexity metrics, Ada 55  
 formal parameters, relationship to actual parameters, Ada 55  
 Format field, Fortran 75  
 Fortran  
     configuration for 74  
     extensions supported 13  
     include files 76  
     macro definitions 76  
     versions supported 13  
 Fortran category, Project Configuration dialog 74  
 frames in windows, sliding 16  
 Frameworks Category, C++ Includes 70  
 free file format, Fortran 75  
 function declarations without parentheses, Fortran 75  
 Function Pointer menu, graphical views 231  
 functions  
     external 230  
     in graphical views 233  
     listing in Filter Area 23  
     static 234  
 fuzzy C/C++ analyzer 66  
 fuzzy search 149
- G**
- gcc build 296  
 General category, Understand Options dialog 94  
 Getting Started dialog  
     displaying at startup 94  
     opening 20  
 Globals menu, graphical views 231

---

Go To Line dialog 168  
Go to Matching Brace option, Search menu 180  
Graph Architecture 195, 201  
Graphic Filter dialog 225  
graphical user interface (GUI), parts of 17  
graphical views 29  
    of architecture 195, 201  
    browsing 224  
    calling methods displayed in 229  
    constants displayed in 230  
    copying to clipboard 236  
    default members displayed in 230  
    diagram of 17  
    entity name truncation for 234  
    expanding or contracting nodes in 225  
    extended by classes displayed in 230  
    extended classes displayed in 230  
    external functions displayed in 230  
    file dependencies displayed in 230  
    filename display options 230  
    filtering 225  
    fullnames displayed in 232  
    function pointers displayed in 231  
    functions displayed in 233  
    globals displayed in 231  
    header files include by's displayed 231  
    hierarchy levels displayed in 232  
    implemented by classes displayed in 231  
    implemented classes displayed in 231  
    imported entities displayed in 231  
    include files displayed in 231  
    inherited entities displayed in 231  
    intrinsic functions displayed in 232  
    invocations displayed in 232  
    layout configuration for 232  
    local items displayed in 232  
    members displayed in 232  
    multiple subtrees displayed in 230  
    objects displayed in 232  
    operators displayed in 232  
    parameters, displaying in 232  
    path highlighting in 226  
    printing 238  
    private members displayed in 233  
    procedures displayed in 233  
    protected members displayed in 233  
    public members displayed in 233  
rename declarations in 233  
reusing 224  
saving 236  
saving as PDF 239  
sorting entities in 234  
spacing entities in 234  
SQL entities shown in 234  
static functions displayed in 234  
synchronizing with other windows 224  
types displayed in 234, 235  
unknown entities, displaying 235  
unresolved entities, displaying 235  
used-by items shown in 235  
uses items shown by 235  
variables displayed in 235  
With By relationships displayed in 235  
With relationships displayed in 235  
    See also structure views; hierarchy views  
Graphical Views menu option 195, 201  
Graphs category, Understand Options dialog 114  
Graphs for option, Graphs menu 201  
Graphs menu 301  
    Dependency Graphs option 201  
    Graphs for option 201  
gray background for text 111  
green text 111  
GUI (graphical user interface), parts of 17  
GVIM editor 113

## H

header files 231  
help 14  
Help menu 304  
    About Understand 14  
    Example Projects option 14  
    Help Content option 14  
    Key Bindings option 14, 182  
    Perl API Documentation option 14, 30, 206  
    Python API Documentation option 14, 30, 206  
    Show Welcome page option 20  
-help option, und command 289  
Hide Inactive Lines option, View menu 180  
hierarchy views 29  
    layout of 232  
    multiple subtrees displayed in 230  
    number of levels in 232  
    parameters, displaying in 232

- See also** graphical views
- Highlight Color**  
in comparison 270
- Highlight Different Words**  
color 270  
in comparison 270
- highlighting full line at cursor 106
- history**  
Information Browser 127, 131  
Source Editor 160  
toolbar for 162
- Horizontal Non-Crossing layout option 232
- HTML**  
exporting metrics to 206  
files 13
- Hyper Grep.** *See* Find in Files dialog
- hyphen (-) collapsing tree in Information Browser 127
- 
- I**
- IB.** *See* Information Browser
- Ignore category, C++ Includes** 70
- Ignore directories in include names field** 69
- Ignore Parent Overrides field** 46
- Implementation fields field, JOVIAL** 80
- Implemented By menu, graphical views** 231
- Implements menu, graphical views** 231
- implicit special member functions, C++** 67
- import files**  
adding to project, Python 85
- imported entities, displaying in graphical views** 231
- importing tool commands** 282
- Imports category**  
Python 85
- Imports menu, graphical views** 231
- importusertools option, understand command** 295
- inactive code, cross-reference information for, C++** 67
- inactive lines, hiding** 180
- include files**  
adding as source files, assembly 59  
adding as source files, C++ 69  
adding as source files, Fortran 76  
adding as source files, PL/M 83  
adding before each project file, C++ 70  
adding in bulk, assembly 59, 61  
adding in bulk, C++ 69  
adding in bulk, Fortran 76
- adding in bulk, PL/M 83  
adding to project, assembly 58  
adding to project, C++ 68  
adding to project, COBOL 61  
adding to project, Fortran 76  
adding to project, PL/M 83  
compiler path for, C++ 66  
displayed in graphical views 231  
environment variables in paths for, assembly 59, 61  
environment variables in paths for, C++ 69  
ignoring during analysis, C++ 70  
Pascal search path 82  
replacement text for, C++ 71  
replacement text for, Fortran 76  
replacement text for, PL/M 83  
system include files, C++ 69
- Include line numbers in rich text field** 108
- Included By menu, graphical views** 231
- Includes category**  
C++ 68  
Fortran 76  
PL/M 83
- Includes menu, graphical views** 231
- Indent field, Editor options** 106
- indentation** 109, 110  
fixing 182
- Information Browser (IB)** 23, 26, 126  
architecture information in 194  
copying information in 126, 131  
displaying from entity in Entity Filter 126  
displaying source of selected entity 129  
entity information displayed by, choosing 128  
expanding and collapsing the tree 127  
for current entity 164  
for current file 164  
history for 127, 131  
location of 17  
metrics in 130, 206  
multiple occurrences open 129  
References in 130  
right-clicking in 22  
saving information in 126, 131  
searching 128  
synchronizing 129
- Information Browser option, View menu** 126
- Inherited By menu, graphical views** 231
- Inherits menu, graphical views** 231

---

initialization files 20  
Inline Function refactoring 174  
Inline Temp refactoring 177  
Insert Spaces Instead of Tabs field, Editor options 106  
installation 14  
Instant Search 148  
Instant Search menu option 148  
interactivity during background processing 95  
interfaces  
    listing in Filter Area 23  
intrinsic functions, parsing, Fortran 75  
Intrinsic menu, graphical views 232  
Intrinsics file field, Fortran 75  
Invocations menu, graphical views 232

## J

.jar files 78  
Java  
    configuration for 77  
    versions supported 13  
Java category, Project Configuration dialog 77  
JavaScript files 13  
JDK, versions supported 13  
JNI external entities, Java 77  
JOVIAL  
    configuration for 79  
    directories for !COPY directives 80  
    versions supported 13  
JOVIAL category, Project Configuration dialog 79  
JPEG format, saving graphical views as 236  
jQuery analysis 87  
Jump to Matching Brace menu option 180  
Jump to Matching Directive menu option 180

## K

Key Bindings category, Understand Options dialog 101, 182  
Key Bindings option, Help menu 14, 182  
Keyboard mappings 101  
Keyboard Scheme field, Key Bindings options 101  
keywords, orange text for 111  
KNI external entities, Java 77

## L

Language auto-architecture 28, 195  
Languages category, Project Configuration dialog 41

-lastproject option, understand command 295  
-lastproject\_cwd option, understand command 295  
Layout menu, graphical views 232  
layout, for graphical views 232  
legacy project  
    conversion 289  
legacy projects, converting 19  
Less memory usages versus speed field, Ada 55  
Level menu, graphical views 232  
libraries, standard  
    Ada, directory for 55  
    including in Analysis Log 103  
    Pascal, paths for 82  
Library directories field, Ada 56  
licensing 14  
line endings for source files 106  
line numbers, displaying in Source Editor 166  
line termination style  
    for saving source files 106  
@lisfile.txt file 294  
Local menu, graphical views 232  
local object declarations, including in project, C++ 67  
local parameters, listed in Entity Locator 24  
Lucene, Apache syntax 148

## M

Macintosh line termination style 106  
macros  
    adding in bulk, C++ 72  
    changing, Ada 57  
    compiler-specific, C++ 66  
    defining in command line 294  
    defining on command line, Ada 57  
    defining, Ada 56  
    defining, C++ 71  
    of editing changes, recording and replaying 183  
    expansion text for, C++ 68  
    importing, Ada 57  
    listing in Filter Area 23  
    recording references when expanding, C++ 67  
    undefined, C++ 72  
Macros category  
    Ada 56  
    C++ 71  
    Fortran 76  
    Pascal 82  
Main subprograms field, Ada 56

- makefile 296  
 Margins field, Editor options 107  
 "Mastering Regular Expressions" (O'Reilly) 159  
**m**  
 members  
     default 230  
     displayed in graphical views 232  
     private 233  
     protected 233  
     public 233  
 Members menu, graphical views 232  
 memory  
     caching include files, C++ 68  
     optimizing analysis to use less, Ada 55  
 menu bar 17  
 menus. *See specific menus*  
 methods  
     listing in Filter Area 23  
 metrics 196, 204, 212  
     displayed in Information Browser 130  
     exporting to CSV file 206, 210  
     exporting to HTML 206  
     for project 207, 208  
     in Information Browser 206  
 Metrics Browser 206  
 Metrics menu 301  
     Browse Metric Charts option 214  
     Browse Metrics option 206, 208  
     Export Metrics option 206, 210  
 Metrics Summary 196, 204, 212  
 Microsoft Visio files, saving graphical views as 236  
 Microsoft Visual C++, as editor 113  
 minus sign (-) collapsing tree in Information Browser 127  
 modules, listing in Filter Area 23  
 multiple users, initialization files for 20
- N**
- Name menu, graphical views 232  
 named root portability mode 53  
 named roots 53  
 Navigation category, Editor options 112  
 Navigation Mode, Source Editor 112  
 nested comments, C++ 66  
 New File option, File menu 183  
 -new option, understand command 295  
 New Project option, File menu 34
- New Project Wizard 97  
 next button 16  
 Next icon 27  
 No Truncation text option 234  
 No Wrap text option 234  
 Node.js analysis 87  
 -noproject option, understand command 295  
 NTFS filesystem 95
- O**
- Objective C/C++ 13  
     enabling 62  
 objects  
     displayed in graphical views 232  
     listing in Project Window 23  
 Objects menu, graphical views 232  
 offline mode 14  
 online help 14  
 Open File option, File menu 183  
 Open last project at startup field, General options 95  
 Open Project option, File menu 19  
 Operators menu, graphical views 232  
 operators, displayed in graphical views 232  
 Options option, Tools menu 93  
 or operators, including in strict complexity, Ada 55  
 orange text 111  
 Overview option, Project menu 207
- P**
- packages  
     listing in Filter Area 23  
 page guide, showing 106  
 Page Setup option, File menu 191  
 parameter declarations without parentheses, Fortran 75  
 parameters  
     cross-reference information for, C++ 67  
     included in listing of 125  
     relationships between formal and actual, Ada 55  
 Parameters menu, graphical views 232  
 parentheses, matching. *See brackets*  
 parse.udb file 17, 33  
 parsing. *See analyzing projects*  
 Pascal  
     configuration for 81  
     macro definitions 82

---

search paths 82  
standard libraries, paths for 82  
versions supported 13  
Pascal category, Project Configuration dialog 81  
PATH definition 287  
path highlighting, in graphical views 226  
pattern matching. *See* regular expressions  
PC line termination style  
    for saving source files 106  
PDF, saving graphical views to 239  
period (.) in regular expressions 158  
Perl  
    API for custom reports 30  
    CodeCheck scripts 255  
    interface for custom reports 206  
    scripts 17  
Perl API Documentation option, Help menu 14, 30, 206  
permission checking 95  
PHP files 13, 87  
PHP version field 87  
pin icons. *See* pushpin icons  
PL/M  
    configuration for 83  
    versions supported 13  
PL/M category, Project Configuration dialog 83  
plus sign (+)  
    expanding tree in Information Browser 127  
    in regular expressions 158  
PNG format, saving graphical views as 236  
Popup Menu category, Tool Configurations dialog 279  
Portability category, Project Configuration 53  
portability mode 52  
portability of project 52  
pound sign (#) comment in command line file 294  
pragma statements, defining macros referenced in, Ada 56  
Predeclared entities file field, Pascal 81  
Prepend the names of externally linkable entities with field  
    C++ 66  
    Fortran 76  
    Pascal 82  
Prepend the names of JNI/KNI external entities with field, Java 77  
preprocessor directives, C++ 71  
Preprocessor field, Ada 55  
preprocessor macros. *See* macros  
preprocessor support, enabling, Fortran 75  
previous button 16  
Previous icon 27  
Print Entity Graph option, File menu 238  
Print option, File menu 191  
printing  
    graphical views 238  
    source files 191, 238  
privacy 14  
Private Members menu, graphical views 233  
procedures  
    in graphical views 233  
programming languages  
    selecting for project 41  
    setting in Source Editor 168  
project 17  
    adding source files on command line 289  
    analyzing (parsing). *See* analyzing projects  
    closing 20  
    closing automatically when opening new project 98  
    configuring 39  
    creating 34  
    creating on command line 287  
    creating, with New Project Wizard 97  
    directory hierarchy for, displaying 132  
    existing, opening 19  
    metrics for 207, 208  
    opening most-recent project at startup 95  
    portability of 52  
    saving configuration of 40  
    storage 32  
    toolbar for 162  
Project Browser 132  
Project Browser option, View menu 132  
Project Configuration dialog 39  
    Ada category 54  
    Assembly category 58  
    C# category 73  
    C++ (Strict) category 63, 73  
    C++ category 66  
    COBOL category 60  
    File Options category 48  
    File Types category 47  
    Files category 42  
    Fortran category 74  
    Java category 77  
    JOVIAL category 79

- Languages category 41
- Pascal category 81
- PL/M category 83
- Python category 84
- saving configuration 40
- Visual Basic category 86
- Web category 86
- Project menu 300
  - Analyze All Files option 88
  - Analyze Changed Files option 88
  - Browse Architectures option 194
  - Configure option 39
  - Overview option 207
- Project Overview 206
- Prompt before closing the current project field, User Interface Alerts options 98
- Prompt for missing include files field 70
- Prompt for missing includes field, C++ Includes 69
- Prompt on parse errors field
  - Ada 56
  - Fortran 76
- \$Prompt variables 273, 274, 275
- Protected Members menu, graphical views 233
- Public Members menu, graphical views 233
- pushpin icons 16
- Python
  - API for custom reports 30
  - configuration for 84
  - versions supported 13
- Python API Documentation option, Help menu 14, 30, 206
- Python category, Project Configuration dialog 84
- Q**
  - question mark (?)
  - in regular expressions 158
  - wild card 158
- quiet\_startup option, understand command 295
- quote ("") prefixing octal constants or string literals 75
- quotes ("""') surrounding normal includes 69
- R**
  - read-only access 48
  - 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 19
  - Record Macro option, Tools menu 183
  - rectangular area, copy and paste 169
  - rectangular text selection 169
  - red project file icon 166
  - refactoring 172
    - Extract Function 175
    - Extract Temp 178
    - Inline Function 174
    - Inline Temp 177
    - Rename 173
  - reference files, C# 73
  - References category, C# 73
  - References, in Information Browser 130
  - regular expressions
    - book about 159
    - in filters for Entity Locator 158
    - in Find dialog 163
  - Reindent Selection command 182
  - relationship 17
  - relative portability mode 52
  - Rename Architecture menu option 196
  - rename declarations
    - displayed in graphical views 233
  - Rename entity 173
  - Renames menu, graphical views 233
  - Replace in Files area 153
  - Replace in Files option, Search menu 153
  - Replacement Text category
    - C++ Includes 71
    - Fortran Includes 76
    - PL/M Includes 83
  - Replay Macro option, Tools menu 183
  - reports
    - customizing 30
    - customizing with Perl or C 206
  - resetlicenseconfig option, understand command 295
  - Right-click Menu
    - external tools available in 274, 279
    - Find in Files option 150
  - right-clicking 16, 122
    - + or - sign in Information Browser tree 127
    - on Analyze icon 88
    - anywhere in Understand 21
    - in Architecture Browser 196
    - bold heading in Information Browser 128

---

creating windows by 21  
on entities to display source 129  
on entities in Entity Locator 24  
on entities in Entity Locator entities 157  
on entities in Information Browser 22  
on entities in Source Editor 21, 27  
in Entity Filter 22  
in Entity Locator 155  
on Entity Locator column headers 156, 158  
in Information Browser 131  
on Selector window 161  
reusing windows by 21, 122  
on selected text 169, 181  
in Source Editor 170  
on Source Editor tabs 185  
    *See also* Ctrl+right-click  
RO (read-only) indicator, in Source Editor 168  
Root filters, Entity Filter 125  
Routines menu, graphical views 233  
routines. *See* functions; procedures  
row colors, alternating 100  
Run Command option, Tools menu 283  
RW (read-write) indicator, in Source Editor 168

## S

Save all modified editor windows when application loses focus field, General options 94  
Save All option, File menu 171  
Save comments associated with entities field  
    Ada 56  
    C 65  
    C++ 67  
    Java 77  
Save duplicate references field, C++ 68  
Save macro expansion text field  
    C++ 68  
Save on command field, User Interface Alerts options 98  
Save on parse field, User Interface Alerts options 98  
Save option, File menu 171  
saving edits automatically 94  
Scope Information, Contextual Information Sidebar 164  
Scope List option, View menu 167  
Scope List, Source Editor 167  
    *See also* Structure Browser, Source Editor 164  
toolbar for 162

scripts  
    CodeCheck 255  
    Perl 17  
Search for include files among project files field, C++ Includes 69  
Search menu 147, 299  
    Find Entity option 155  
    Find in Files option 25, 150  
    Find option 163  
    Go to Matching Brace option 180  
    Instant Search option 148  
    Replace in Files option 153  
Search Paths Category, Pascal 82  
Search Results window 25  
searching  
    files 147, 150  
    graphical views 223  
Select All option, Edit menu 169  
Select Block menu option 180  
Selector area 161  
sharp sign (#) comment in command line file 294  
shortcut commands 101  
Show Page Guide field, Editor options 106  
Show standard library files field, Analyze options 103  
Show tabs field, User Interface options 96  
Show the Getting Started dialog on startup field, General options 94  
Show Welcome Page option, Help menu 20  
SlickEdit editor 113  
sliding frames 16  
-SlowConnect option, understand command 295  
Soft Wrap option, View menu 182  
Sort menu, graphical views 234  
Sort Selection command 182  
sorting entities 124  
Sound beep field 98  
Sound beep on analysis completion field, Analyze options 103  
Source Editor 27, 166  
    auto-complete options 109  
    auto-indent options 109, 110  
    bookmarks, creating 184  
    bookmarks, navigating 184  
    bracket matching in 180  
    Browse Mode in 169  
    case, changing in 181  
    closing files 171

- colors in, customizing 111, 166  
 colors in, default 111  
 commenting and uncommenting code 182  
 configuring 105  
 Contextual Information Sidebar (CIS) in 164  
 copying to clipboard from 169  
 creating files 183  
 displaying by right-clicking on entities 129  
 displaying from Find Results window 25  
 displaying from Search Results window 152  
 external editor replacing 113  
 folding (hiding blocks in) code 180  
 hiding inactive lines in 180  
 history of locations visited in, moving through 160  
 jumping to specific line number 168  
 keystrokes in, list of 182  
 language, setting 168  
 line numbers displayed in 166  
 list of specific structures in 164, 167  
 location of 17  
 macros of editing changes, recording and replaying 183  
 moving between windows of 27  
 opening files 183  
 printing from 107, 191  
 read-write (RW) and read-only (RO) indicators 168  
 right-clicking in 21, 27, 170  
 right-clicking tabs in 185  
 saving files 171  
 Scope List in 167  
 searching and replacing text in source files 163  
 searching in source files 147  
 status bar in 168  
 status icon in 166  
 tabs in, controlling behavior of 185  
 toolbar for 162  
**source files**  
 adding to project 42, 133  
 adding to project on command line 289  
 analyzing using projects. *See* project  
 closing 159, 171  
 creating 183  
 deleting from project 44  
 directories for, adding to project 43  
 directories for, deleting from project 44  
 displaying by double-clicking entity 129  
 displaying by right-clicking on entities 129  
 displaying from Find Results window 25  
 displaying from Search Results window 152  
 editing. *See* Source Editor  
 encoding for 48, 106  
 excluding from source list 44  
 external editor for 113  
 include files specified as, assembly 59  
 include files specified as, C++ 69  
 include files specified as, Fortran 76  
 include files specified as, PL/M 83  
 line endings for 106  
 list of, generating from command line 294  
 listing in Filter Area 23  
 moving between windows of 27  
 opening 183  
 overriding settings for 46  
 portability of 52  
 printing 191, 238  
 printing, configuration for 107  
 read-only access for 48  
 removing from project 133  
 saving 171  
 searching 25, 147, 150  
**Sources tab, Project Configuration dialog** 42  
**spaces, converting tabs to** 106  
**Spacing menu, graphical views** 234  
**special member functions, C++** 67  
**split**  
 Source Editor 181  
 workspace 160  
 workspace, toolbar for 162  
**Sql menu, graphical views** 234  
**SQL, embedded, in Pascal** 81  
**square brackets. *See* brackets**  
**src.jar file** 78  
**src.zip file** 78  
**Standard field, Ada** 55  
**standard libraries**  
 Ada, directory for 55  
 displaying in Analysis Log 103  
 Pascal, paths for 82  
**Standard Library Paths category, Pascal** 82  
**standards, CodeCheck** 243  
**Start menu, Understand commands in** 19  
**startup**  
 Getting Started dialog displayed on 94  
**static functions, displayed in graphical views** 234

- 
- Static menu, graphical views 234  
statistical usage reporting 14  
status bar, Source Editor 168  
status icon, in Source Editor 166  
status line 17  
Sticky box 160  
strict C/C++ analyzer 63, 73  
strict complexity, count and/or operators in, Ada 55  
Structure Browser, Contextual Information Sidebar 164  
structure views 29  
    *See also* declaration views; graphical views  
Styles category, Editor options 111  
subprograms, listing in Filter Area 23  
subtraction sign (-) collapsing tree in Information  
    Browser 127  
support contact information 13  
SVG format, saving graphical views as 236  
switches, command line  
    understand command 295  
synchronizing Information Browser 129  
Sysroot field, C++ Includes 69  
system include files, C++ 69
- T**
- tab, automatic 109, 110  
tabs for windows, displaying 96  
tabs, converting to spaces 106  
technical support contact information 13  
temporary bookmark 184  
text  
    comparing 263  
    copying to clipboard 169  
    selecting 169  
Text Comparison window 263  
text favorite 145  
Text menu, graphical views 234  
TextPad editor 113  
title areas, filenames in 97  
title bar 16  
Title Formats field, User Interface options 97  
Toggle Overtype option, Edit menu 182  
tool commands, importing and exporting 282  
Tool Configurations dialog  
    Popup Menu category 279  
    Toolbar category 281  
toolbar  
external tools available from 281  
hiding and displaying icons 162  
location of 17  
visibility of icons in, controlling 281  
Toolbar category, Tool Configurations dialog 281  
Tools menu 302  
    Editor Macros option 183  
    Options option 93  
    Run Command option 283  
    User Tools option 272, 274, 280, 282  
tools, external. *See* external tools  
tooltips 112  
Treat system includes as user includes field  
    C++ Includes 69  
Tree row indentation field, User Interface options 100  
treemap  
    CodeCheck 249  
    metrics 213  
Truncate column field  
    Fortran 75  
    JOVIAL 79  
Truncate Long text option 234  
Truncate Medium text option 234  
Truncate Short text option 234  
truncation at column  
    Fortran 75  
    JOVIAL 79  
Turbo Pascal 81  
types  
    displayed in graphical views 234, 235  
    listing in Filter Area 23  
Types menu, graphical views 234  
Typetext menu, graphical views 235
- U**
- .udb file extension 17, 33  
UML Sequence Diagram 221  
Uncomment Selection menu option 182  
und command 286, 287  
    adding files to project 289  
    analyzing a project 293  
    options in latest version, listing 289  
undefined macros, C++ 72  
Undefines category, C++ Macros 72  
Understand  
    compared to compiler 32  
    contact information 13

features of 12  
 multiple users for 20  
 online help for 14  
 starting 19  
 starting from command line 295  
 windows in 16, 17  
 understand command 295  
**Understand Options dialog** 93  
 Analyze category 103  
 Dependency category 104  
 Editor category 105  
 General category 94  
 Graphs category 114  
 Key Bindings category 101, 182  
 User Interface category 96  
 undocking windows 16  
 Unicode file handling, in comparison 270  
**Unix**  
 line termination style, for saving source files 106  
 unknown entities, displaying in graphical views 235  
**Unknown menu, graphical views** 235  
 unresolved entities  
     displaying in graphical views 235  
**Unresolved menu, graphical views** 235  
 unresolved variables, listed in Entity Locator 24  
**Update Information Browser field, Browse Mode Editor**  
     options 112  
**uperl command** 17  
**upython command** 17  
 usage reporting 14  
**Use alternating row colors field, User Interface options**  
     100  
**Use case-insensitive lookup for includes field**  
     C++ Includes 69  
**Use default working directory field, General options** 95  
**Use include cache field, C++** 68  
**Use preprocessor field, Fortran** 75  
**Use the New Project Wizard when creating new projects field, Configure options** 97  
**Usedby Menu, graphical views** 235  
**User Interface category, Understand Options dialog** 96  
 user interface, parts of 17  
**User Tools option, Tools menu** 272, 274, 280, 282  
 users, multiple, initialization files for 20  
**Uses Menu, graphical views** 235

**V**

**variables**  
     displayed in graphical views 235  
     listed in Entity Locator 24  
     unresolved 24  
**Variables menu, graphical views** 235  
**Version field**  
     Ada 54  
     Fortran 74  
     Java 77  
     JOVIAL 79  
     Pascal 81  
**vertical bar (|) in regular expressions** 159  
**Vertical Non-Crossing layout option** 232  
**VHDL**  
     terminology 85  
     versions supported 13  
**vi editor** 113  
**View menu** 300  
     Analysis Log option 89  
     Bookmarks option 161, 184  
     Browse Mode option 169  
     Contextual Information option 164  
     Entity Locator option 24, 155  
     Favorites option 25, 143  
     Hide Inactive Lines option 180  
     Information Browser 126  
     Project Browser option 132  
     Scope List option 167  
     Soft Wrap option 182  
     Window Selector option 161  
**views.** See **declaration views; graphical views; hierarchical views**  
**violations**  
     checking for 243  
**Visio files, saving graphical views as** 236  
**-visit option, understand command** 295  
**Visit Source field, Browse Mode Editor options** 112  
**Visual Basic**  
     configuration for 86  
**Visual Basic category, Project Configuration dialog** 86  
**Visual C++, as editor** 113  
**Visual Studio**  
     as external editor 113  
     auto-architecture 195

---

## W

-wait option, understand command 295  
warnings, displaying from Analysis Log window 89  
watched directories  
    setting 44, 45, 46  
Web category, Project Configuration dialog 86  
Web support 13  
websites  
    external editor information 113  
    O'Reilly and Associates 159  
Welcome page 19  
white file icon 166  
whitespace  
    in comparison 270  
    indicators for 107  
Whitespace field, Editor options 107  
wild cards, in filters for Entity Locator 158  
Window menu 159, 303  
    Close All Document Windows option 159, 171  
    Close option 159, 171  
    Windows Navigator option 160  
Window Selector option, View menu 161  
windows 16, 17  
    animated opening and closing of 96  
    closing 16  
    creating with Ctrl+right-click 21, 122  
    docking and undocking 16  
    filenames in title area of 97  
    frames in, sliding 16  
    list of 123  
    list of open windows 160, 161  
    organizing 159  
    reusing for graphical views 224  
    reusing with right-click 21, 122  
    tabs for, displaying 96  
Windows category, Application Styles options 100  
Windows category, User Interface options 99  
Windows line termination style  
    for saving source files 106  
Windows Navigator option, Window menu 160  
With Bys menu, graphical views 235  
Withs menu, graphical views 235  
working directory, default 95  
Wrap Long text option 234  
Wrap Medium text option 234  
Wrap Mode field, Advanced Editor options 108

Wrap Short text option 234  
wrapping lines  
    display 182  
    printing 108

## X

XML output 139

## Y

yellow project file icon 166